Distributed and Parallel Database Systems

Divya Yadamreddi
dyadamre@asu.edu
Arizona State University

Harshitha Katpally
hkatpall@asu.edu
Arizona State University

Pragna Munagala
pmunagal@asu.edu
Arizona State University

ABSTRACT

This paper introduces the concept of spatial data and the tools available out there to efficiently manage and process such huge amounts of data. It also highlights the challenges faced while handling this data and the need for such tools. Among the different categories of data, one category that this paper focuses on is the Geospatial data. Like every other category, this one also spreads out to large scales in quantity and hence the need to handle it arises. One of the many tools available to organize and manage geospatial data is the GeoSpark software which is considered to be a very efficient and systematic utility in its task. This paper describes the essentials of GeoSpark, the phases undergone to perform computations on data using this software, the experimental results and learning experience from this project.

KEYWORDS

ApacheSpark, GeoSpark, Hadoop, MapReduce, Hadoop Distributed File System.

1 INTRODUCTION

The enormous increase in the size of data over the past few years has made it difficult to handle and process it in an efficient and faster way. Different types of data needs different techniques to handle it. In this project we have worked on managing geospatial data. Spatial data or geospatial data is a special kind of data which has a geographical aspect to it. This means the data is described in terms of location on the earth. Generally, the data includes, address of a location, coordinates and size of an object on the planet, etc. There are two basic forms of spatial data. The vector form which uses points, lines and polygons to represent the spatial features and the raster form which uses cells to represent these features where single cells can be used to represent cities, linear sequence of cells can be used to represent roads. Geospatial data is being used widely in many applications in the recent times. It can be used to search for restaurants within a particular distance range, to provide road maps and to locate the number of users of a particular application in a particular region. With regards to the size of the planet one can imagine the size of spatial data and the challenges faced in handling such huge amounts of data. Hence, developing and using special methods to handle geospatial data is very essential.

Distributed and parallel systems have been proven to be suitable to manage this kinds of huge data in a parallel and scalable manner by taking advantage of the system architectures. In aim of this project is to understand the importance of distributed computing and learning the features of the paradigm to optimize the processing of data.

The project has been divided into three main phases to learn distributed computing in an orderly manner.

1.1 Phase 1

This phase mainly focused on setting up the environment for Hadoop clusters and playing around with the Hadoop interface to make work easy in the later phases of the project. A 3-node Hadoop cluster with 1 master and 2 slave nodes was setup and few basic geospatial queries have been applied on a distributed spatial dataset. It was focused on learning the fundamental partitioning techniques such as R-tree index, equal grid and a combination of both each of which have their own advantages in query processing.

1.2 Phase 2

In this phase, two user defined functions have to be created, namely ST_Contains and ST_Within using which various Spatial Queries had to be performed. The functions had to be implemented in SparkSQL using Scala which has to perform four different queries, two in each of the functions. ST_Contains function was used to conduct the Range Query where a query rectangle R and a set of points P where given as input and all the points within the given R had to be calculated and returned. The Range Join Query was also performed in ST_Contains, where a set of rectangles R and a set of points S was the input to the function and it had to return all pairs of (Point, Rectangle) sets such that the point is within the rectangle. Whereas the ST_Within function had to perform Distance Query to which a point P and a distance D were the inputs and it had to return all the set of points which lie within a distance D from P. Also, the ST_Within had to perform the Distance Join Query to which a set of points S1, a set of points S2 and a distance D were passed as arguments and it had to perform some calculations and return all pairs of (s1, s2) such that s1 is within a distance of D from s2(here s1 belongs to S1 and s2 belongs to S2). This experiment was tested on the Thoth Lab cluster and the results were submitted to Vocareum which is a self-grading environment.

1.3 Phase 3

This phase focused on conducting a spatial hot spot analysis on a spatiotemporal big data in order to identify statistically significant spatial hotspots using Apache Spark. This analysis included two tasks. One is the hot zone analysis in which a Range Query had to be performed on a rectangle dataset and a point dataset. The rectangle which includes more points within it among the given points is said to be hotter when compared to a rectangle containing less number of points. So, this method would return the hotness of each of the rectangles. The other analysis that was performed is the hot cell analysis which had to calculate the Getis-Ord statistics on a collection of New York City Yellow Cab taxi trip datasets. This method had to determine the top 50 hot spots and they are returned as sorted by their G-score in a descending order. This phase was aimed to get deeper insights into handling clusters and outliers in spatial data and identifying them using some spatial statistics. In

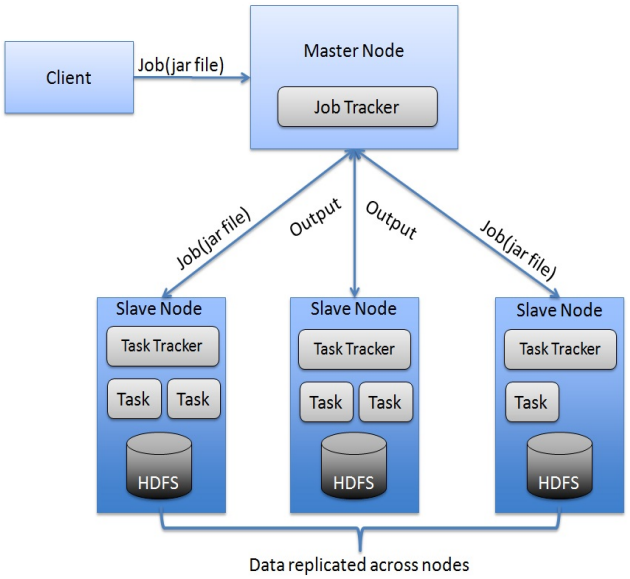

Figure 1: Hadoop high level architecture.

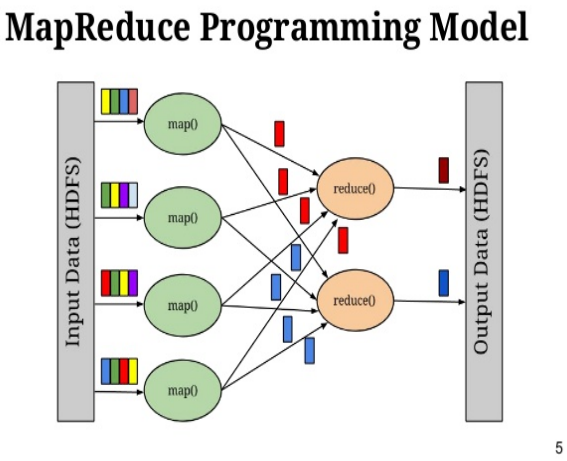

Figure 2: MapReduce operation.

Getis-Ord statistics, z-score and p-values are provided which help in understanding the formation of clusters. This phase also had to be submitted on Vocareum similar to phase 2 and the scores were obtained automatically.

2 SYSTEM ARCHITECTURE

There are three main building blocks of the system architecture, namely, Hadoop layer, Apache Spark layer and GeoSpark layer. Substantial effort is required to set up each of the environments separately and later to connect them. A detailed description of the importance of these components and how they are setup is given.

2.1 Hadoop Layer

Hadoop is an open-source software framework for storing data and working on them on clusters of hardware components. It enables easy storage of massive amounts of data and high speed processing power in order to process the data and to parallely perform multiple tasks on different pieces of the data.

Hadoop comprises of four major modules, each of which have specific tasks to perform in handling applications on the cluster environment. The Hadoop system uses its own file system called the Hadoop distributed file system (HDFS) which stores data in a easily accessible format across a large number of linked storage devices. The processing functionality of Hadoop is handled by the MapReduce mechanism, where the data is processed in two phases, namely, map and reduce in the same order. The map phase essentially maps the data into a suitable format in which it can be handled efficiently. The reduce phase, actually reduces data into the desired result by processing it and operating on it. Two other modules of Hadoop are Hadoop Common which provides the necessary tools for the operating systems to handle the HDFS and Yarn which stores the resources of this system that is performing the analysis on data. The Hadoop framework operates with two types of nodes. One is the master node which controls the functioning of other nodes named worker nodes i.e. it assigns the tasks to be performed to each of the worker nodes and contains the information about all of them. Whereas the worker nodes just perform the tasks assigned to them by the master.

The MapReduce framework is composed of three operations. The Map phase where each of the worker nodes applies the map function to the local data and writes output to the temporary storage. The next phase is shuffle, where the worker nodes redistribute data based on output keys so that all data with the same key are present in one worker node. The final phase is reduce, where the worker nodes process each group of output data in parallel to produce the output.

The file storage system used in Hadoop is one of the most efficient storage systems which facilitates high availability by replicating data at multiple nodes, reliability on data again due to replication, fault tolerance as a crash of one of the sites would not result in loss of data completely since that data will be present in other nodes and scalable to adding more clusters to the system. Every file is divided into segments called blocks. The default block size is 64MB which can be increased if necessary. HDFS is also cost-efficient as it is built on low-cost hardware components.

2.2 Apache Spark Layer

Apache Spark is considered as the largest open-source project that supports distributed data processing due to its elegant APIs and interfaces for performing operations on such data. It is capable of handling very large amounts of data which is located on several clusters of machines. It has a rich set of libraries and APIs to support many languages like Java, Python, Scala, and R. Apache Spark runs the applications in-memory as opposed to the Hadoop's MapReduce where data is processed to and from computer hard drives[1]. Spark

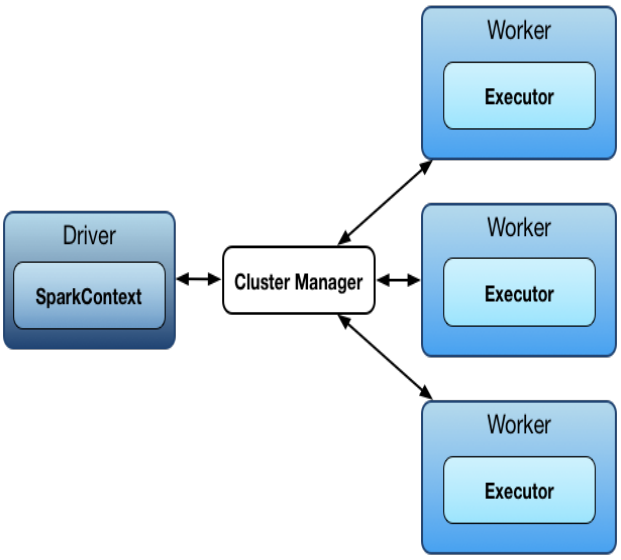

Figure 3: Apache Spark architecture.

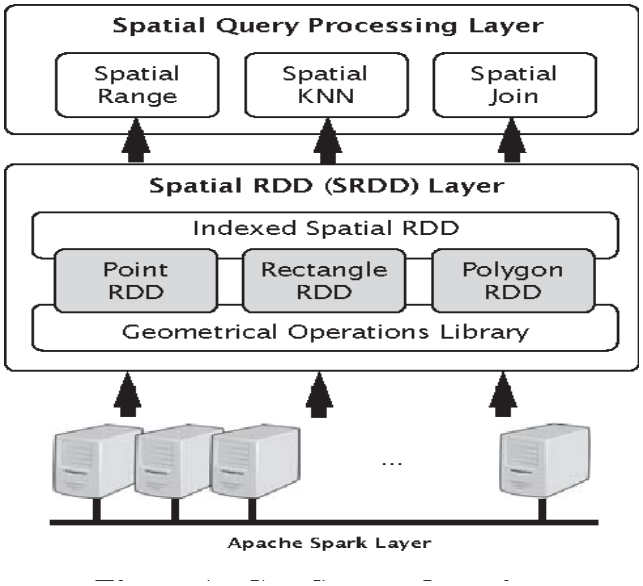

Figure 4: GeoSpark architecture layers.

is also using along with Hadoop's HDFS, which is why Apache Spark has been used in this project.

Since large amounts of workloads cannot be handled by using Spark on a single machine, it will normally run on big data clusters i.e. Hadoop. The Hadoop's YARN resource manager will manage the Hadoop cluster as well as Spark and can be used to compute Hadoop jobs. The concept of Resilient Distributed Dataset (RDD) is often discussed in the context of Spark[2]. This is a logical in-memory read only dataset distributed across a cluster to incorporate fault-tolerant and efficient management of data in Hadoop.

2.3 GeoSpark Layer

GeoSpark is a fully-fledged in-memory cluster computing framework that allows easy loading, processing and analyzing large clusters of spatial data in Apache Spark. It is specially designed to support and manage spatial distributed data by transforming the RDD to SRDD (Spatial RDD) to store data. GeoSpark provides a geometric operations library that can access Spatial Resilient Distributed Datasets (SRDDs) to perform basic geometric operations on them. It also contains a rich set of APIs to support spatial queries such as Spatial Range, KNN, Join Queries on large scale datasets.

The GeoSpark layer comprises of three main layers, namely Apache Spark Layer, Spatial RDD Layer and the Spatial Query Processing Layer. Basic spark functionalities such as loading and storing data along with regular RDD operations are performed by the Apache Spark Layer. Spatial RDD Layer consists of three Spatial RDDs which extend regular Apache Spark RDD in order to support geometrical and spatial objects. Finally, the Spatial Query Processing Layer executes spatial query processing algorithms such as spatial range, spatial KNN, etc on SRDDs very efficiently. GeoSpark also has a special ability to decide whether a spatial index needs to be created locally on an SRDD partition at run time to maximize memory and CPU utilization in the cluster and also improve its run time performance.

Only three of the spatial query processing algorithms have been used in the project. They are explained in detail.

2.3.1 Spatial Range Query. The Spatial Range Query queries over a specified range i.e. a rectangular query window to find all the geometric points within that range. Building indexes on the Spatial RDD and then performing this query would affect the execution time and aid is faster execution. First we load the dataset, partition it and build R-Tree index on it [3]. The query is then forwarded to all the partitions, is applied on each of the datasets and the results are obtained.

Steps Involved:

(1) Define an envelope based on the input.
(2) Read the dataset from the HDFS that is uploaded and convert the PointRDD to ObjectRDd.
(3) Execute the Spatial Range Query on the dataset. GeoSpark method to do this is SpatialRangeQuery().
(4) Return count of the number of points that are returned.
(5) Repeat this process on the dataset after building an R-Tree Index on it.

2.3.2 Spatial KNN Query. KNN stands for K-Nearest Neighbors. As the name suggests, this query finds k-neighbor points that are nearest to a spatial point.To do this, GeoSpark uses a heap based top-k algorithm. This query consists of performing two operations i.e. selection and merge. A Spatial RDD operation is performed on the input dataset and this query is done on each of the partitions separately. To each partition a point P and a number k is given as input. During the selection phase, the distance between each of the points in the partition and the point P is calculated and the result is maintained in a local heap. This heap always contains the top k-nearest points to point P[3]. Similar to Range Query, indexing can

be performed on the partitioned data to improve the execution time of the query. This is followed by the merge phase where the top k-nearest neighbors from all the partitions are merged to get the k-nearest neighbors to point P in the entire dataset. The memory and CPU utilization can be increased by performing R-Tree indexing on the partitioned dataset.

Steps Involved:

(1) Create a point from the coordinates of the input provided which is the point to which k-nearest neighbors have to be calculated.
(2) Read the dataset from the HDFS that is uploaded and convert the PointRDD to ObjectRDd.
(3) Execute the Spatial KNN Query on the dataset. GeoSpark method to do this is SpatialKNNQuery().
(4) The coordinates of the k-nearest neighbors to the input point P are calculated and returned as output.
(5) Repeat this process on the dataset after building an R-Tree Index on it.

2.3.3 Spatial Join Query. The Spatial Join Query performs a join operation on a set of RectangleRDD which represents a rectangle in space and a PointRDD which is a single geometrical point. First, GeoSpark will create partitions on the input dataset. Then, the Spatial Join is performed based on grid IDs. If the algorithms finds two same grid IDs then the join is performed , else if the objects overlap, then both the objects are kept in the result[3]. Finally, the algorithm will check for any duplicates in the result, eliminate them and return the result. Similar to Spatial Range and Spatial KNN Queries, indexing the partitioned datasets improves the execution time of the query.

Steps Involved:

(1) Define an envelope based on the input.
(2) Read the dataset from the HDFS that is uploaded and load it into a PointRDD.
(3) Execute the Spatial Join Query on the dataset. GeoSpark method to do this is SpatialJoinQuery(sc, PointRDD, RectangleRDD).
(4) Return the size of the resultant dataset.
(5) Repeat this process on the dataset after building an R-Tree Index on it.

3 PROJECT

3.1 Phase 1

Phase 1 includes setting up of cluster using Hadoop and Apache Spark. The Hadoop cluster consists of three nodes - one master node and two slave nodes. Hadoop, as explained above, is an open source Apache Project that allows creation of parallel processing application on large data-sets that are distributed across networked nodes [4]. It also has a Hadoop Distributed File System to handle redundancy and scalability of data across nodes. It has a Hadoop YARN which is a framework for job scheduling that executes data processing tasks on all nodes.

While configuring, a Secure Shell is used, to provide a secure channel over an unsecured network. The Secure Shell is nothing but a connection between an SSH client and an SSH server. These configurations are duplicated in the master as well as slave nodes. The HDFS in Hadoop is used to store the data files on every node. To execute Geospatial queries using Geo Spark APIs, we have deployed GeoSpark jar in the Apache Spark environment.

Tasks Performed:

(1) The first step was to create Point RDD or Geospatial RDD.
(2) The second step was to implement Spatial Range query by querying the Point RDD in the query window.
(3) The third step was to implement Spatial KNN query. Here R tree was built indexing on Point RDD and the nearest five neighbors were found.
(4) The fourth step was to implement Spatial Join query. The rectangle RDD and point RDD are joined using R tree with and without using the index.

Connection between the the nodes is established using SSH. Start-all.sh and stop-all.sh commands were used to start and stop the clusters in Hadoop.

3.1.1 Steps to Configure SSH.

(1) Public and private keys are generated on both the slave nodes.
(2) The public keys of both the slaves are copied to the master.
(3) The public keys are then authorized by adding them to the list of authorized keys in both the slave nodes.
(4) Connect to the master by using the password on the slave nodes first and then log out of the shell.
(5) Login to the master using only the IP address.

3.1.2 Steps to Configure Hadoop.

(1) Since the master should be able to connect to the slave nodes, the IP addresses of all the nodes, master and the two slaves, are added to the host file on master.
(2) The slave nodes only need to be connected to the master and do not need any communication among themselves. Hence the IP address of the master node is added to the host file of slaves.
(3) The path of JAVA folder is added to the Hadoop environment file on all the three nodes in the cluster.
(4) The property tags in coresite.xml, mapred-site.xml and hdfs-site.xml are appended in all the three nodes.
(5) The Hadoop namenode is run as "hadoop namenode -format" to format the file system. Run the command start-all.sh to start the Hadoop system.

3.1.3 Steps to Configure Apache Spark.

(1) To the spark environment file of the master node, add the IP address of the master.
(2) To begin Apache spark, run the command start-master.sh from the master node.
(3) To establish a connection between the slaves and master, run the script spark-class org.apache.spark.deploy.worker.

3.2 Phase 2

Phase 2 included the task to execute spatial queries on a data set of New York City Yellow cab Taxi trip. The programming language, Scala was used to implement user defined functions - ST_Contains and ST_Within in SparkSQL. There were four queries to be executed

- Range query, range join query, distance query and distance join query.

Range query: This is executed by implementing the user-defined function ST_Contains - using a query rectangle R, the sets of point within the rectangle are returned.

Range join query: This is also executed by implementing the user-defined function ST_Contains - when a set of rectangles and a set of points are given, the tuples [points, rectangle] are returned where the point is within the rectangle.

Distance query: This is executed by implementing the user-defined function ST_Within - when a location P and distance D are given, the points that are within the distance of D from P are returned.

Distance join query: This is also executed by implementing the user-defined function ST_Within - if a set of points S1 and S2 and a distance D is given, the sets [a,b] are returned such that a lies within D distance from b with a belonging to set S1 and b belonging to set S2.

The data files, in the format of csv, are loaded into DataFrame using csv format by the given template in Scala code.

The user defined functions ST_Contains and ST_Within have two input parameters and a single output parameter.

Inputs: ST_Contains - pointString : String, queryRectangle : String. ST_Within - pointString1 : String, pointString2 : String, queryRectangle : String

Outputs: ST_Contains - Boolean (true or false) ST_Within - Boolean (true or false)

Functionality: ST_Contains - given a rectangle and point query, returns if the point is fully contained within the rectangle or not.

ST_Within - given two points and a distance value, it returns if two points are within the given distance or not.

The code was first tested on the static IP machines after setting up Hadoop, Apache Spark and Geo Spark. The command spark-submit was used to submit the jar file to Vocareum. A larger data set is tested on the same code in Vocareum.

3.2.1 *Steps to implement and run Phase 2.*

(1) First, we run the Hadoop cluster by invoking master. We then run invoke Spark environment.
(2) After changing the source code for the user defined functions, ST_Contains and ST_Within from the template in Github consisting of required functionality, the respective jar file is generated.
(3) The jar file is generated by running the command - sbt clean assembly, in the folder where build.sbt is located.
(4) After submitting the input files - arealm.csv and zcta510.csv to the Hadoop File system, submit the jar file and generate the output using the setup cluster.
(5) The output is generated as a csv file which can be downloaded and verified.

This phase was necessary for the project to help us understand the working and implementation of Scala environment and programming in Scala. It helped us get comfortable using Scala to code the required functionalities with better efficiencies when bigger data-sets were involved.

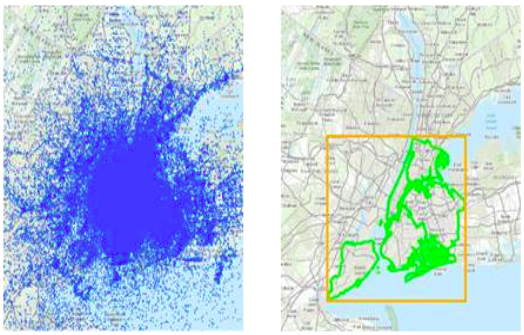

Figure 5: Dataset of New York Taxi Trip. [5]

3.3 Phase 3

This phase required us to perform spatial hotspot analysis. We needed to complete two different hot spot analysis tasks. A data set consisting of Yellow cab taxi trip data for New York City for the year 2009-2015 was provided. The dataset included the information about pickup time, date, drop-off time, date, the latitude and longitude of the location, trip distance, fare and the passenger count. We were required to find 50 most significant hot-spots (hot cells and hot zones) from the data given to us.

Here, the input is taken as pick up and drop locations and joined with the latitude and longitude of the hot zone. The distance at which the point lies nearest to the hot zone determines the multitude of the zone. This information is combined with the complete information given to that particular zone. This is the procedure to calculate or determine the hot zone.

The metric, Gettis-Ord score (z-score) which is provided by ACM SIGSPATIAL is used for the hot cell analysis for each cell location.

3.3.1 Hot zone. Analysis - In this task, we need to perform a range join operation on a rectangle data set and a point data set. For each rectangle, the number of points located within the rectangle are returned. The hotter the rectangle, more the points are present in that rectangle. This task requires to calculate the hotness of all the rectangles.

Implementation -

(1) Load both the point data and the rectangle information.
(2) A utility function is implemented to check if the point lies in the given rectangle or not.
(3) The set of all points that lie within the rectangle are then combined and the count determines the hotness of that particular rectangle.
(4) After calculating the hotness of all the rectangles, the entire data set is ordered on the basis of the degree of hotness and is returned as the output.
(5) The output is generated as a csv file.

3.3.2 Hot Cell. Analysis - This task will focus on applying spatial statistics to spatiotemporal big data in order to identify statistically significant spatial hot spots using Apache Spark.

The input to this problem is the data set of collection of New York city Yellow cab taxi reports from January 2009 to June 2015.

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

Figure 6: Formula for Getis-Ord Statistic. [5]

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

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

Figure 7: Mean and Variance Formulae. [5]

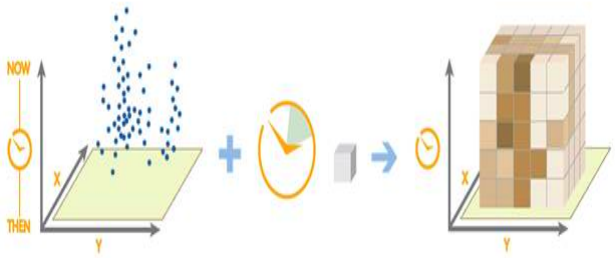

Figure 8: Algorithm in a nutshell. [5]

The source data is clipped to an envelope encompassing the five New York City boroughs in order to remove some noisy error data. The output is a list of fifty most significant hot spot cells in time and space as identified using the Getis-Ord statistic.

Where xj is the attribute value for cell j, wi, j is the spatial weight between cell i and j, n is equal to the total number of cells and

The neighborhood for each cell in the space-time cube is established by the neighbors in a grid based on subdividing latitude and longitude uniformly. This spatial neighborhood is created for the preceding, current, and following time periods. For simplicity of computation, the weight of each neighbor cell is presumed to be equal.

Time and space should be aggregated into cube cells as specified on command lines -

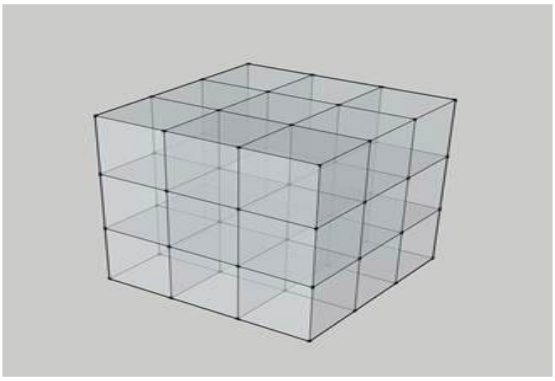

Figure 9: The 26 neighbors of a cell in the 3-D space with the cell itself also its neighbor. [5]

Using spatial statistics to identify statistically significant clusters or outliers in spatial data is a well understood analytic approach employed by GIS professionals. Spatial statistics is required when making decisions - e.g., if we are 95 % sure that the levels measured in this area exceed a regulatory threshold, then we must respond by doing such and such. When identifying statistically significant clusters (often termed Hot Spot Analysis), a very commonly used statistic is the Getis-Ord statistic. It provide a z-score and p-values that allow users to determine where features with either high or low values are clustered spatially. It is important to note that this statistic can be applied to both the spatial and spatiotemporal domains; in this competition, we are focused on the spatiotemporal Getis-Ord statistic.

The following changes were made to reduce the computation power need -

(1) The input will be a monthly taxi trip dataset from 2009 - 2012. For example, "yellow_tripdata_2009-01_point.csv", "yellow_tripdata_2010-02_point.csv".
(2) Each cell unit size is 0.01 * 0.01 in terms of latitude and longitude degrees.
(3) You should use 1 day as the Time Step size. The first day of a month is step 1. Every month has 31 days.
(4) You only need to consider Pick-up Location.
(5) Jaccard similarity is not used to check the answer.

Implementation -

(1) The taxi trip information is loaded.
(2) After validating the points, the valid points are kept as per the information provided about the square.
(3) Gi* statistics are found using the spark SQL queries and the utilities by using parameter values.
(4) The neighbors are calculated as per the geometry of the cube.
(5) The output is then generated as z-score after counting all parameters and other parameters.
(6) As per the z-score, the data is sorted to get the top hot cells.
(7) Then the top 50 top hot cells are generated and output as csv file.
(8) The output file can be downloaded and verified for correctness.

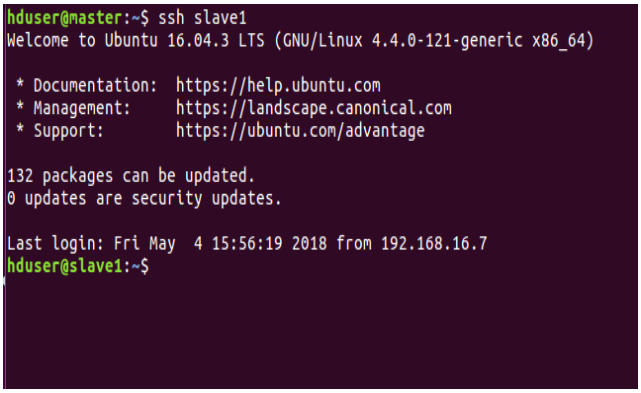

Figure 10: Password less ssh from master to slave1.

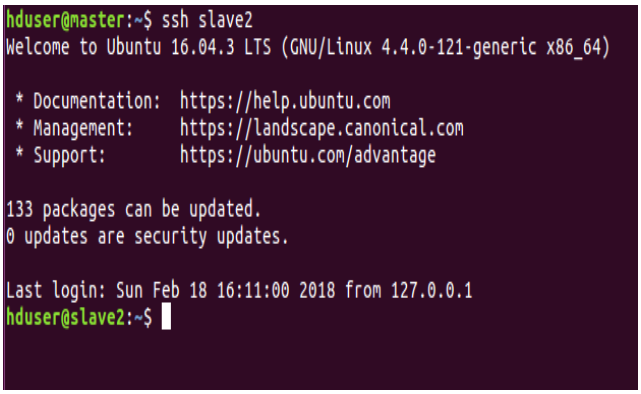

Figure 11: Password less ssh from master to slave2.

(9) The command sbt clean assembly was used to build the code.

4 RESULTS AND ANALYSIS

The results obtained in various phases of the project are mentioned in the following sections.

4.1 Phase 1

The setting up of Hadoop cluster configuration with one Master and two Slave nodes and starting the cluster and spatial queries had been performed.

The master and slave nodes are configured as password less ssh. Master can ssh into slave1 without password as shown in figure 10:

Master can ssh into slave2 without password as shown in figure 11:

The same way either of the slaves can do password less ssh to master for example slave1 as shown in figure 12:

The Spark UI displaying the workers i.e., two slave nodes is as shown in figure 13:

4.2 Phase 2

Sample output for a given point and rectangle performing Range Query, Range Join Query, Distance Query and Distance Join Query

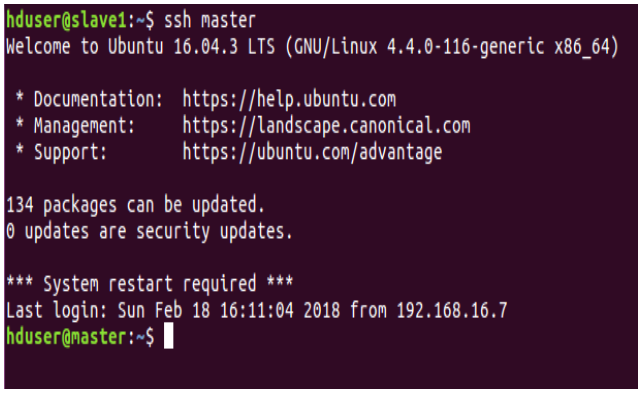

Figure 12: Password less ssh from slave to master.

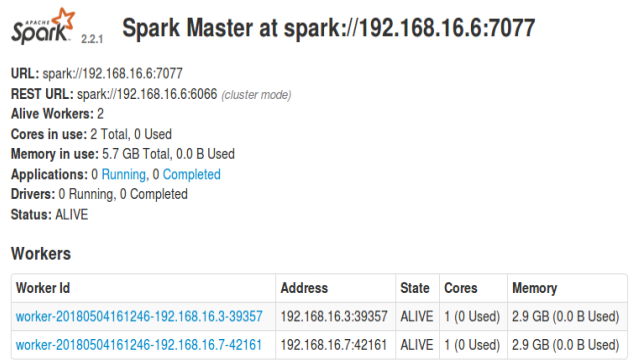

Figure 13: Spark web UI.

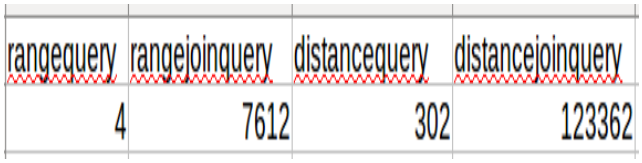

Figure 14: Phase 2 Output.

using the implemented ST_Contains and ST_Within is as shown in figure 14:

Where, Range query: Given a rectangle and a set of points, returns the number of points within the rectangle. Range join query: Given a rectangle and a set of points, returns the number of point, rectangle pairs such that the point is fully contained within the rectangle. In Range query and Range join query, ST_Contains is used in this implementation to find out if a point is fully contained within the rectangle or not. Distance query: Given a point and distance in km, returns the number of points which lie within the given distance from given point. Distance join query: Given a set of points S1, S2 and distance in km, returns all (s1,s2) pairs such that the s1 and s2 are separated by the given distance. S1 belong to S1 and s2 belongs to S2.

Rectangle Coordinates	Hotness
-73.789411,40.666459,-73.756364,40.680494	1
-73.793638,40.710719,-73.752336,40.730202	1
-73.795658,40.743334,-73.753772,40.779114	1
-73.796512,40.722355,-73.756699,40.745784	1
-73.797297,40.738291,-73.775740,40.770411	1
-73.805770,40.666526,-73.772204,40.690003	3

Figure 15: Hot Zone Analysis Output.

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	29
-7398	4075	28
-7398	4075	14
-7399	4074	15
-7398	4075	22
-7399	4075	27
-7399	4074	23
-7398	4075	30
-7398	4075	16
-7399	4074	22
-7399	4074	16
-7398	4076	15

Figure 16: Hot Cell Analysis Output.

4.3 Phase 3

The output file holding the rectangle data and its hotness sorted based on hotness (number of points enclosed within rectangle) is as shown in figure 15:

The output file holding the top 50 results of hot cells based on Getis-Ord statistic is as shown in figure 16:

In the output shown above, x indicates latitude, y indicates longitude and z indicates day.

The jar files of phase 2 and phase 3 implementations are uploaded on Vocareum, a self grading system which grades the submission by running it against various test cases in the configured cluster.

5 LEARNING EXPERIENCE

The purpose of the whole project was mainly to help visualize the cost of handling large amounts of data and and the need for distributing this data through the use of distributed database systems. Along with the distributing of data, this project helped to learn the parallel execution of operations on this data and how this will increase the execution time of operations on large-scale data. It also helped in understanding GeoSpark, which is a software designed to perform distribution of data and parallel execution of operations on it.

Phase 1: During this phase the basic installation and setup of GeoSpark along with setting up the Hadoop cluster and connecting Hadoop and GeoSpark has been learned and completed. All the steps required to install Hadoop and develop a cluster of systems that communicate with each other through a password less shh link. Also, implementing basic Spatial Queries on distributed Spatial data in GeoSpark was learnt.

Phase 2: The main learning exposure during phase 2 was getting hands on experience with using Scala, implementing distributed database systems and queries in Scala. The way in which data can be loaded into HDFS, is distributed across clusters, computations distributed to clusters and implemented on each cluster of data using Hadoop has been clearly reviewed in this phase. Also, it helped learn executing the most essential Spark Query Algorithms on GeoSpark. This phase helped in getting ready for phase 3 which involved developing a larger application by operating on a very large-scale data.

Phase 3: This is the final phase of the whole project and involved a higher level of computation on Spatial data. By working on a real-time spatial dataset and organizing it into the Hadoop cluster increased the level of understanding in managing such kinds of huge data. Secondly, making sense of how the data is actually stored at the database level was essential. Finally, the steps involved in executing operations on distributed data using GeoSpark on the real-time data was completely understood.

The overall objective of working on clusters of data and efficiently operating on them has been well-understood and analyzed.

6 CONCLUSION

Distributed Databases are something that the current era of technology requires strongly to deal with the great amount of data that is being added to every day. To deal with the huge amount of data, we require special technologies like Hadoop and HDFS to efficiently make use of data and learn to use it to its fullest capacity. This project has helped us to achieve that goal. During the course of this project, we have learnt to use different technologies like Hadoop and Spark and also learnt to communicate with databases with languages like Scala. We were able to calculate the different hot zones of the New York city based on a data set given to us. We were able to manipulate and understand the data to provide the required results. We were also able to study the data and return outputs for different sets of data and different scenarios.

7 ACKNOWLEDGEMENT

We are grateful to Prof. Mohamed Sarwat for sharing his sharing his pearls of wisdom with us during the course of this project. We would

also like to thank our TA, Yuhan Sun for guiding us throughout the project and providing his inputs towards the successful completion of the same.

REFERENCES

[1] MAPR-Getting Started With Apache Spark. *.https://mapr.com/ebooks/spark/03-apache-spark-architecture-overview.html*
[2] MAPR-Spark 101: What is it, What it does, and Why it matters. https://mapr.com/blog/spark-101-what-it-what-it-does-and-why-it-matters/
[3] Jia Yu, Jinxuan Wu, Mohamed Sarwat. GeoSpark: A Cluster Computing Framework for Processing Spatial Data
[4] Linode-How to Install and Set Up a 3-Node Hadoop Cluster. https://linode.com/docs/databases/hadoop/how-to-install-and-set-up-hadoop-cluster/
[5] ACM SigSpatial Cup 2016. http://sigspatial2016.sigspatial.org/giscup2016/problem