Geo-spatial Data Analysis with SparkSQL

Abhilash Kumar Chaudhary, Prem Patel, Sumit Rawat
Arizona State University, University Drive, Tempe AZ 85281
{achaud32, pdpate10, srawat7}@asu.edu

Abstract

As the economy of the world is growing the observational data related to that economic business increases. Now the data collected can be of any form or any type and this data can be found very useful to make strategies for increasing the market profit in future and also to mitigate the mistakes that was performed in past. According to Guting [3] spatio-temporal is very useful observational data which consists of datasets like vehicle tracking data, GPS devices in mobile phones, moving hurricane in different regions, spread of forest fires, spread of epidemic diseases etcetera. This data provides us the insight regarding the maximum probability of happening of that event in certain regions using this spatial data. We have given a task to perform analysis on certain data given by a New York based taxi firm. We created an environment with a distributed setup of master and slave computing machine, where the master will control all the tasks and distribute it to slaves. Furthermore, we used the Google Cloud Computing instances to create this architecture. As the data set was huge and unstructured we used Apache Spark and HDFS (Hadoop File System) to perform the analysis as the task can be performed faster and with good efficiency.

1. Introduction

In this faster growing world data is generated everywhere in every format one can imagine. One such type of data format is Geo-Spatial Data. According to Fishman [2], Geo-Spatial data is any data with a spatial identifier referring to a position on the earth: a house, building, road, lake, mountain, or countless others. Data like weather reports, suggested routes on Google Maps, Geo-tagged Tweets, Store Locations etc are an integral part of our daily lives and it falls into the category of the Geo-spatial Data. Also Geo-spatial data is a big influence on today's market and the businesses that incorporate Geo-spatial data into their analysis, forecasting and reporting have the potential to grow their business exponentially through smarter use of this kind of data. The Geo-spatial data is captured using the latitude and longitude of the location. Spatial analysis can be used through a Geographic Information System (GIS) to determine optimal site locations, detect clusters or pattern of events by which we can make predictions and understand events occurring around the places. The implementation of our project lies around this idea of finding patterns of events occurring at certain places.

In our project we are given the data of some New York based taxi firm where we are given the task of performing and run multiple spatial queries on this data. The data consists of the real time locations of the customers. The data is created everyday and even one can say every minute throughout the entire year and this data will never stop from being created. So using the traditional system to run the analysis on this kind of big data will take forever to analyze. So we have to come up with a system which can analyze terabytes of data in some polynomial time. Here is where the Distributed Database Systems come handy. In this system the analysis is not performed using the traditional system using only the one machine. In the distributed system there are two or more systems working together to perform the analysis of the data, where one system will be Master node and all the other remaining nodes are Slave nodes. The Master node controls all the slave nodes and assign some tasks to perform to each slave node. This way the work is divided to all the systems and we can get the results faster than the traditional system.Here, since all the data is unstructured and we have to use the SparkSQL and Big data systems to perform the analysis. We will use spatial queries to perform the operations on the data. These type of queries are different from the traditional SQL queries where it cannot allow the use of points, lines and polygons. Using the simple geometric concepts we try to find the patterns that is present in the data-set. This analysis we call as Hot Spot analysis. In this Hot Spot analysis we have two types of analysis Hot Zone analysis and Hot Cell analysis. In the former we try to find the zone which has the maximum number of passengers during the time frame. In hot cell analysis we try to find the space-time cells with the maximum z-scores. These are further explained in the upcoming sections.

2. Methodology

In order to work with spatio-temporal data, an environment of virtual machines was created. The details of the environment are discussed here:

2.1. System Architecture

Before performing large-scale geo-spatial data analysis on huge data, an environment needs to be build up to perform all kinds of analysis. To setup such environment, the major requirement is to have an environment that is reliable, has data replication, has fault tolerance and scalable on need basis. Such environments comprise of virtual machines that will achieve enhanced performance and cost effective computation. Google Cloud E2 instances provide all these properties along with security and monitoring.

2.1.1 Infrastructure

With Google Cloud E2 instances, it is possible to perform data analysis on huge data-set with low computational cost and high performance while maintaining the network connections throughout the environment. The environment designed initially consists of 3 nodes of VM instances and then scaled to 5 nodes of VM instances to compare the results.

The 3-node cluster is controlled by one master node ("cluster-ac5c-m") and two worker nodes ("cluster-ac5c-w-0" and "cluster-ac5c-w-1") in the same network. It is crucial to achieve bidirectional password-less SSH (as shown in Figure 1 and Figure 2 below) among all the nodes to maintain free communication throughout the cluster.

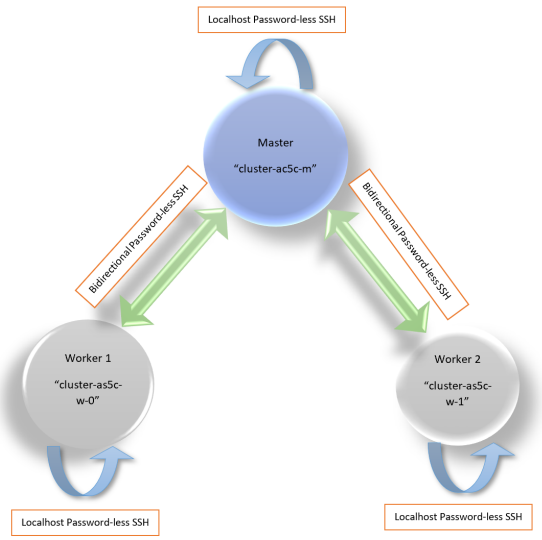

Figure 1. Google Cloud E2 Instance Cluster

This is achieved by generating keys using SSH-keygen among all nodes and adding "id_rsa.pub" keys in "authorized_keys" on all machines.

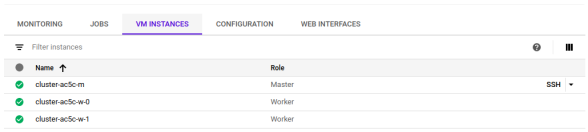

Figure 2. E2 VM instances

2.1.2 Apache Hadoop and Apache Spark

Apache Hadoop makes use of the Google Cloud E2 VM Instances' master-slave network to perform data analysis and computation on humongous data-sets by use of Hadoop Distributed File System (HDFS) and the MapReduce functionality. Apache Hadoop fulfils the data replication and data integrity requirement of cluster. Hadoop is installed on all the VM Instances of the cluster with proper configuration to set up master-slave relationship in "core-ste.xml" and "slaves" file. The hadoop cluster and data-nodes can be monitored in web browser "master_ip:50070" as shown in Figure 3.

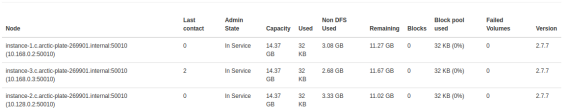

Figure 3. Hadoop Cluster

To perform data analysis operations on huge datasets, data parallelism and fault tolerance proves helpful. The in-memory cluster computing framework of Apache Spark enables the users to compute on entire clusters of Resilient Distributed Dataset (RDD). Along with Apache Hadoop, Spark is also installed on all the nodes. Hadoop and Spark are started only on master nodes. Before starting hadoop, HDFS should be formatted by running "hadoop namenode -format" and then started by "sbin/start-dfs.sh". Spark is started by "sbin/start-all.sh". The Spark servers can be checked on web browser "master_ip:8080" as shown in Figure 4.

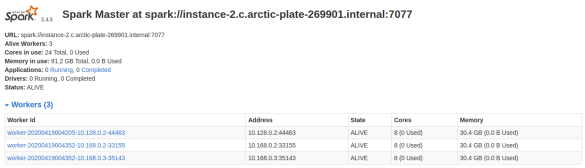

Figure 4. Spark Cluster

After setting up the environment, the project was completed in three phases. The details of these phases are discussed below:

2.2. Phase 1

In the first phase, four spatial queries were run on it using SparkSQL. These queries were run on a data-set consisting of 10,000 points and 10,000 rectangles. The data-set for the points consisted of their x and y coordinates whereas the data-set for the rectangles consisted of a pair of x and y coordinates for the bottom-left and the top-right vertices. As defined in the Github repo[5], the queries consisted of:

1. **Range Query** that returns all the points that lie within the given rectangle coordinates.
2. **Range-Join Query** that returns all the Point-Rectangle pairs where the point lies within the corresponding rectangle in the pair.
3. **Distance Query** that returns all the points that lie within the given distance radius from a given point.
4. **Distance-Join Query** that returns all the Point-Point pairs where the first point lies within the given distance radius from the second point.

To run these queries, two functions were defined in Scala, namely **ST_Contains** and **ST_Within**.

- **ST_Contains**: It takes the coordinates of a point and a rectangle as arguments and returns whether the point lies in the rectangle. This is achieved by comparing the coordinates of the point with the coordinates of the diagonal of the rectangle. The point is judged to be inside the rectangle if its coordinates are less than the corresponding minimum and maximum coordinates of the rectangle. This function was used to implement the Range Query and the Range-Join Query.
- **ST_Within**: It takes the coordinates of two points and a distance value as arguments and returns whether the points lies within the provided distance value. This is achieved by calculating the square of the euclidean distance between the points and comparing it to the square of the distance value. This function returns true if square of the euclidean distance between the points is less than the square of the distance value. This function was used to implement the Distance Query and the Distance-Join Query.

2.3. Phase 2

The second phase of this project builds on the developments of the first phase. This phase implements hot spot analysis on spatio-temporal big-data with the help of two tasks. The data-set used consists of the passenger pick-up location points from a collection of New York City taxi trip data from 2009 to 2012. The data is cropped to include only the five New York City boroughs (latitude range 40.5N to 40.9N, longitude range 73.7W to 74.25W) for the removal of noise. The size of a cell is 0.01 * 0.01 in terms of the latitude and longitude degrees and a time step is defined as a day starting from the first day of the month and assuming that all the months have 31 days. As defined in the Github repo[6], the tasks implemented in this phase are as follows:

1. **Hot Zone Analysis** - The hotness of a rectangle(cell) is defined by the number for points that lie within it. This tasks aims to calculate the hotness of every rectangle(cell) provided in the data-set and return the results in an order sorted by the rectangle's coordinates. To calculate the hotness of a rectangle(cell) we perform the Range-Join operation, implemented in phase 1, on the rectangle and points data-set.
2. **Hot Cell Analysis** - In this task we rank the significance of the rectangles(cells) by ordering them on the basis of their Getis-Ord statistic value. Here, for each cell in the data-set, we define a space-time cube, as defined in the page [1], by including its adjacent cells in space for the past, present and future time step. In such a space-time setting, a cell has 26 neighbors and we consider that all these neighbors have the same cell weight. The Z scores obtained by calculating the **Getis-Ord statistic** show where the high and low value features cluster in space. It views each cell within the context of its adjacent cells as a feature with a high value is interesting but may not be a statistically significant hot spot for the data. A statistically significant hot spot will itself have a high value and will also be surrounded by other cells with high values as well. Ord and Getis[4] calculate the G^*_i statistic for the cells using equation 1:

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

in 1 x_j is the attribute value for cell j, $w_{i,j}$ is the spatial weight between cell i and j, and n is the total number of cells, and the other variables are defined in equations 2 and 3:

$$\overline{X} = \frac{\sum_{j=1}^{n} x_j}{n} \quad (2)$$

and

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

This task then returns the fifty most significant cells in time and space in the descending order of their Getis-Ord statistic value.

2.4. Phase 3

The third phase of this project evaluates the environment by running different sizes of data-sets on varying number of machines. Two types of virtual machine setup was used in the experiments:

- **1 master 2 slaves setup**: 1 Google Cloud E2 instance functions as a master node and 2 others as slave nodes.
- **1 master 4 slaves setup**: 1 Google Cloud E2 instance functions as a master node and 4 others as slave nodes.

Two sets of data were used in the experiments:

- A small sample Data-set consisting of 100,000 tuples (17 MB size) of taxi trip data.
- A large Data-set consisting of taxi trip data (2.8 GB size) for the month of October 2009.

The parameters on which these runs were evaluated are described as follows:

1. **Memory Usage** - The amount of memory being used by the nodes during the run of the tests.
2. **Incoming Network traffic** - The amount of data (in bytes) received by a node via the network during the run of the tests.
3. **Outgoing Network traffic** - The amount of data (in bytes) sent by a node via the network during the run of the tests.
4. **CPU utilization** - The percentage of CPU resources used by the node during the run of the tests.
5. **Disk utilization** - The amount of data (in bytes) read from the disk on the node during the run of the tests.
6. **Total run time** - The total run-time of the tests.

To collect the data for these parameters, a bash script was used that writes the values into a log file during the run of the experiments. The findings of this phase are discussed in the next section.

3. Experimental Evaluation

To evaluate the experiments, a bash script file was created for to log the values of the parameters. The memory usage of the node is logged by using the **free** command, CPU utilization using the **top** command, disk utilization using the **df** command and the incoming and outgoing network traffic using the **iftop** command as can be seen in figure 5.

```
#! /bin/bash
printf "Memory\t\tDisk\t\tCPU\t\tSend Rate\t\tReceive Rate\t\tCurrent Time\n"
end=$((SECONDS+3600))
while [ $SECONDS -lt $end ]; do
MEMORY=$(free -m | awk 'NR==2{printf "%.2f%%\t\t", $3*100/$2 }')
DISK=$(df -h | awk '$NF=="/"{printf "%s\t\t", $5}')
CPU=$(top -bn1 | grep load | awk '{printf "%.2f%%\t\t\n", $(NF-2)}')
SEND_RATE=$(sudo iftop -t -s 5 -N 2>/dev/null | awk '/Total send rate/ {printf "%s\t\t\t",  $6}')
RECEIVE_RATE=$(sudo iftop -t -s 5 -n -N 2>/dev/null | awk '/Total receive rate/ {printf "%s\t\t\t", $6}')
now=$(date +"%T")
echo "$MEMORY$DISK$CPU$SEND_RATE$RECEIVE_RATE$now"
done
```

Figure 5. Bash script for data collection

The script repeatedly collects the values of these parameters and logs them in a txt file along with the timestamp of the readings as can be seen in figure 6.

```
abhilash@cluster-974e-m:~/CSE512-Project-Phase2-Template-master$ ./run.sh
Memory          Disk            CPU             Send Rate               Receive Rate            Current Time
53.92%          9%              0.14%           7.38Kb                  16.5Kb                  19:24:47
53.92%          9%              0.10%           44.2Kb                  17.4Kb                  19:25:01
53.92%          9%              0.09%           8.71Kb                  26.0Kb                  19:25:15
53.92%          9%              0.07%           44.1Kb                  17.1Kb                  19:25:30
53.91%          9%              0.05%           6.36Kb                  15.1Kb                  19:25:44
53.91%          9%              0.04%           44.1Kb                  18.4Kb                  19:25:58
53.91%          9%              0.10%           8.29Kb                  19.0Kb                  19:26:13
53.90%          9%              0.08%           7.01Kb                  22.7Kb                  19:26:27
53.90%          9%              0.06%           7.05Kb                  18.3Kb                  19:26:41
53.90%          9%              0.05%           8.96Kb                  14.6Kb                  19:26:56
53.92%          9%              0.04%           6.32Kb                  20.0Kb                  19:27:10
53.92%          9%              0.03%           8.62Kb                  25.3Kb                  19:27:24
53.96%          9%              0.39%           6.81Kb                  26.2Kb                  19:27:39
53.99%          9%              0.30%           8.47Kb                  18.6Kb                  19:27:53
53.99%          9%              0.24%           7.08Kb                  16.6Kb                  19:28:07
```

Figure 6. Output of the bash script

In this section we will do certain experiments by changing certain parameters and observe the fluctuations in the result. The parameters that we are going to experiment with are changing the number of slave nodes working under the master node and changing the size of the datasets. The results which we are measuring and comparing here are runtime of the program and communication cost on slave nodes.

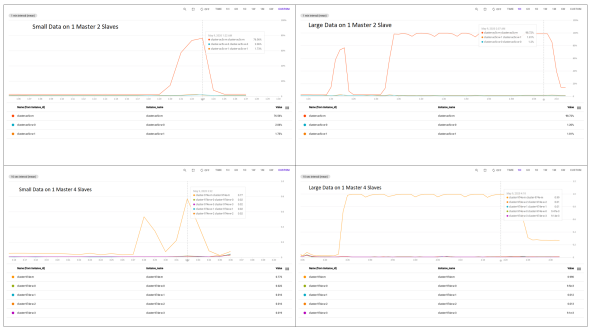

Figure 7. CPU Utilization evaluation

In 7 we have plotted the CPU utilization Graph running against the worker nodes showing the usage of the CPU during the different scenarios.

At the beginning we have one master node and two slave nodes for which we measure the results of run-time and communication cost. Also we are using a large data-set for this experiment. The results can be as seen in figure below:

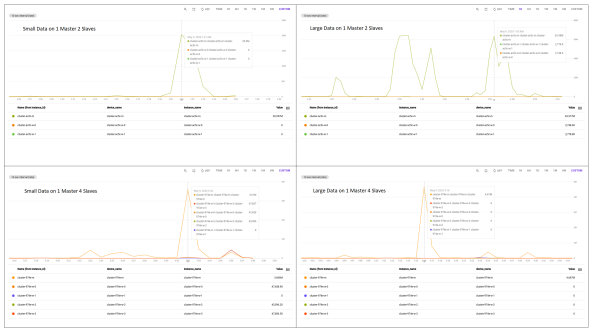

Figure 8. Disk Read evaluation

Disk Read Evaluation graph 8 gives us the knowledge of how the disk read bytes increase when the dataset size is increased. However, the disk read bytes decrease when the number of nodes are increased in the cluster.

Figure 9. Memory Usage

The memory usage is the prime feature which shows us how the program is running and in our case we can see this trend which is graphically represented in figure 9 as how the memory usage in the master node is almost same but the load in shared among the worker nodes.

Figure 10. Received Bytes

The graph shown in figure 10 shows the spikes in the received bytes when using the large datasets and more number of worker nodes.

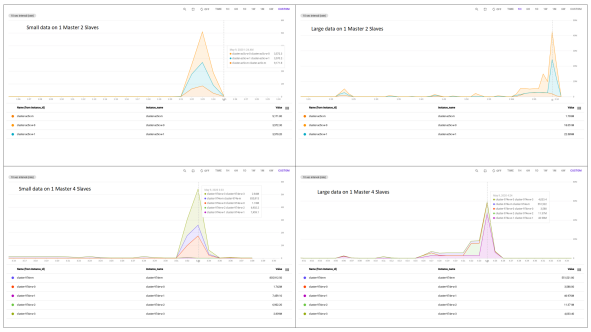

Figure 11. Sent Bytes

Figure 11 graph doesn't show any different results than received traffic. The incoming traffic increases with the size of the dataset and when the number of nodes are increased in the cluster

Cluster Size / *Dataset Size*	***1 Master 2 Slaves***	***1 Master 4 Slaves***
Small	92 seconds	87 seconds
Large	2404 seconds	2200 seconds

Figure 12. Total runtime

The run time analysis for the experiments can be found in the figure 12. The run time of a test was calculated by comparing the system time during the start and completion of the scala program. The figure shows that the runtime decreases when more nodes are used in the cluster.

4. Conclusion

At the end of our project the results we found were intriguing and interesting. The experiments provided insightful knowledge about the working of the distributed systems. HDFS (Hadoop File System) and Apache SparkSQL are promising when the task involves data analysis computations on big dataset. The experiments aim to observe the fluctuation in results on varying various parameters i.e. either change the configuration of the cluster or change the data size. The computations were done on small and large dataset on 3-node clusters and 5-node clusters. The results from these experiments show that increasing the instances in the cluster would decrease the running time of the program. CPU utilization increased with the size of the dataset. Network traffic (*incoming* and *outgoing*) increased with the size of the dataset as well as with the number of the nodes in the cluster. Disk read bytes increase with the size of the dataset but decrease with the increase in number of nodes in the cluster. Memory usage was similar on the master nodes in all the cases, however, the worker nodes shared the load.

The future aspect of this work would certainly include more variations in the configurations of the cluster and the variety of data.

References

[1] ACM. ACM SIGSPATIAL Cup 2016. http://sigspatial2016.sigspatial.org/giscup2016/problem.

[2] Jamie Fishman. What is geospatial data—and why should businesses care about it? https://blog.westmonroepartners.com/what-is-geospatial-data-and-why-should-businesses-care-about-it/.

[3] Ralf Hartmut Güting. Spatio-temporal data types, 2009.

[4] Keith Ord and Arthur Getis. Local spatial autocorrelation statistics: Distributional issues and an application. *Geographical Analysis*, 27:286 – 306, 09 2010.

[5] YuhanSun. CSE512-Project-Phase1-Template. https://github.com/YuhanSun/CSE512-Project-Phase1-Template.

[6] YuhanSun. CSE512-Project-Phase2-Template. https://github.com/YuhanSun/CSE512-Project-Phase2-Template.