Spatial Hot Spot Analysis using Apache Spark

Tarun Chinmai Sekar
tsekar@asu.edu-1213181929
Arizona State University
Tempe, AZ

Sneha Lakshminarasimhan
slaksh29@asu.edu-1215126471
Arizona State University
Tempe, AZ

Siddarth Madan Kumar
skuma147@asu.edu-1215148025
Arizona State University
Tempe, AZ

ABSTRACT

The Hot Spot Analysis aims at statistically discerning atypical patterns in the available dataset. The project hence recognizes the 'Hot Spots' from the Spatio Temporal data based on the New York City's Yellow Taxi trip records with the help of Getis-Ord Statistic. This report provides insights into the environmental setup for Hadoop and Apache Spark used in the project and their implementation specifics comprising of user-defined functions for the Spatial Hot Spot Analysis performed here.

KEYWORDS

Hadoop, HDFS, Apache Spark, Scala, MapReduce, GetisOrd, Hotspots, RDD, Large-Scale data, Spatial Analysis, DAG, Spark UDF, Hot Zone, Hot Cell

1 INTRODUCTION

Every day the amount of data being collected through the innumerable smart devices keeps increasing exponentially. The volume of the data incubating devices has become untameable and so is the data collected. The vastness of this data provides great insights into the people and the vehicles, which has been identified to be helpful in understanding the patterns of their mobility to provide ameliorated infrastructure and services. The myriad spatiotemporal data being collected, shows great trends, including trajectories, tweets, and mobile Twitter users, etc., allows us in analyzing potential and profound patterns as well as non-trivial insights.

Among these patterns, we observed the spatial hotspot, which had objects within the zone showing a higher degree of positive correlation as compared to those that are outside. The observation and analysis of the results have great virtue to be used in various applications of a lot of domains. The spatial statistics like Getis-Ord are useful in determining the degree of correlation between the points. As the size of the information keeps increasing, the quality of this analysis shows improvement while the processing of data becomes cumbersome. The difficulty lies in developing efficient techniques for the sake of processing the data.

The underlying data on which the project is based in the New York City Taxi and Limousine Commission Yellow Cab trip data which contains the information about the taxi trips around the city of New York. Every record of the trip taken in the city consists of attributes like pickup and drop-off times and locations, fare amount, passenger count, and distance traveled. The demographic setup is as follows. To obtain the statistical value, space has been discretized by the use of the uniform spatial grid. Time is split into steps. This creates a spatiotemporal cell. For each spatiotemporal cell, the counts of the passengers are aggregated.

Figure 1: Spatio-Temporal cube

A neighborhood of a cell is that which is adjacent to it through an edge or the vertex, which carries equal weight. The data set is publicly available to exploit for the critical usefulness and worthiness it could provide the public.

2 PROBLEM DEFINITION

2.1 Phase 1

During Phase 1, 3 AWS instances were created, in which 1 served as master and the other 2 as slaves after Hadoop was installed.[] This was followed by posting a video on youtube regarding the Hadoop setup and configuring the Spark Cluster. Implemented a method ST_Contains and ST_Within in SparkSQL and utilized them to perform spatial queries.

2.2 Phase 2

We are provided with a large collection of New York City Yellow Cab taxi trip records spanning January 2009 to June 2015, clipped to encompass the five New York City boroughs.

We define a cab trip as:

$$J = (loc\_start, t\_start, loc\_end, t\_end, p)$$

$$where$$

$$loc\_start = (lat\_start, long\_start)$$

are the start point latitude and longitude and

$$t_{start}$$

is the time stamp of start-time, p is the number of passengers on the trip respectively.
Saptio-Temporal Cell [4] is defined as the cell
c = (id,p,s)
where
id = (x,y,t) , p is the passenger count, s is the score of the cell respectively

A HotSpot can be viewed as a cell with the maximum number of passengers dropped off inside each of the cells, which we can calculate by summing the passenger counts of all the journeys having the drop-off points inside the respective cell.

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

where

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

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

Getis-Ord [5] Statistic is used to calculate the score of each cell, that is the Z-score and thus no calculations are required.

The above equation illustrates that the score of each cell is dependent on its neighbouring cells and itself. As stated in problem statement [4], all the cells have equal weights and the size of the neighbourhood of a cell is 1 cell + 26 adjacent cells = 27 cells. The output is the list of most significant hot spot cells in time and space.

3 SYSTEM ARCHITECTURE

3.1 Hadoop And MapReduce

The Apache Hadoop framework is used to execute large-scale cluster computing and can scale vertically from a single server to thousands of machines within a few seconds where each server have it is on computing power and storage. Two types of services are provided - distributed storage and processing for big data. The storage service is called as Hadoop Distributed File System - HDFS and MapReduce [6]

MapReduce is a parallel programming model used for architecture that processes large amounts of data on clusters in parallel.

3.2 Apache Spark

The goal of the Spark project was to make it more efficient and easy to user with the added benefits of MapReduce is scalable, distributed, fault tolerant processing framework.

Spark Application running on a cluster:

- A Spark application is run as an individual process coordinated by SparkSession object.
- The resource or cluster manager maps tasks to workers, one task per partition.
- The role of the task is to apply its computation on the partition so as to give a new partition as resultant dataset.
- Efficiency is increased through caching as iterative algorithms apply the same operations.
- The computed results are sent back to to the driver application or be saved to the disk.

Spark supports the following cluster managers:

[8]Spark Standalone - a simple one machine cluster manager included with Spark Apache Mesos - a general cluster manager that can also run Hadoop applications Apache Hadoop YARN - the resource manager in Hadoop 2 Kubernetes - an open source system that automates deployment,scaling

3.3 GeoSpark

GeoSpark is an extension of Apache Spark/SparkSQL which provides Spatial Resilient Distributed Datasets (SRDDs) / spatial SQL that works efficiently on spatial data across the machines. [7]
It consists of three modules:

- Spatial RDD
- Spatial SQL
- Complex geometries/ trajectories: point, polygon.

3.4 Spark User Defined Functions

User Defined Function helps in creating new Column Based functions that add the capability to Spark for transformation on datasets. UDFs are great when built-in SQL functions are not up to this task. It accepts scala functions of up to 10 input parameters. UDFs are used in both Phase 1 and 2.

3.5 Ganglia

Ganglia is an open source project and once enabled on cluster, the performance against different metrics can be viewed and its a scalable, distributed system. [1]

4 ENVIRONMENTAL SETUP

4.1 Preliminary Setup

Setup Platform :

- AWS Elastic Compute Cloud (EC2)
- Operating System - Linux Ubuntu 18.04 LTS 64-bit
- AWS EC2 Instance type
  For initial Setup(Phase 1 & 2):
  - Master - t2.medium
  - Slaves - t2.medium
  For Experimental Setup (Phase 3):
  - Master - t2.large
  - Slaves - t2.large
- Number of cores per node -
  - Master(t2.large) - 2
  - Slaves(t2.large) - 2
  - Slaves(t2.medium) - 2
- Size of each node:
  - Medium - 4GB
  - Large - 8GB

- Java OpenJDK version - 1.8.0
- Apache Hadoop - 2.7.7
- Apache Spark - 2.3.2

(1) Create 3 EC2 instances, and designate as 1 Master and 2 as Slaves
(2) Install Java 1.8.0 version

(3) Setup Password-less Log-in using Bidirectional SSH

5 EXPERIMENTAL SETUP

5.1 Phase 1

There are 2 CSV files fed as input for this phase:

- area1m1000.csv: Contains list points in 2d space.
- zctal10000.csv: It contains a list of pairs of points in 2D space representing rectangle diagonals. [2]

The task is to implement 2 UDF ST_Contains and ST_within in Spark SQL used to perform four spatial queries. Range query: This makes use of ST_Contains. Find all the points within R where R is the query rectangle.

Given a query with rectangle R which is coordinates string of diagonally opposite points of rectangle and P, set of points. Initially, we load into the point data frame, all the points and then run a SQL select query on it to implement custom predicate evaluation written in ST_Contains to determine if the specified points lie in the rectangle. If evaluation results into True, then the point will be considered. The set of points that lie in the rectangle is obtained as a result.

Range join query: This outputs all pairs(Point, Rectangle) such that the point is within the rectangle.

5.1.1 ST_CONTAINS. Compute the maximum of the points X, maximum y, minimum X and minimum y coordinates for each pair of points from the given coordinate rectangle diagonal pair points. We check for the condition that the specified points lie between minimum x and maximum x, and the same goes for point 'y' if it lies between minimum y and maximum y, the point lies within the rectangle and function returns true, else returns false.

5.1.2 ST_WITHIN. The Euclidean distance between the coordinates of pair of points is computed using the below formula:

$$calc\_Distance = Math.sqrt(Math.pow((pt1\_x - pt2\_x), 2) + Math.pow((pt1\_y - pt2\_y), 2))$$

The distance calculated from the above formula is compared with given distance D. If its found to be lesser than D, the function returns True, else return False. After running all the specified queries, the CPU utilization was found to be 34% and wait utilization was on average 13%.

5.2 PHASE 2

In this phase, Geo-spatial analytics were performed to develop SparkSQL solution. This consists of doing Hot zone analysis and hot cell analysis. [3]

5.2.1 Hot Zone Analysis. In this task, we are provided with rectangles and points in the '.csv' file. Range join operation was performed on each of the rectangles. Hot cells are identified as the rectangles where the most number of points are found within them. Then a Cartesian product of the points and the points of the rectangle is performed to retrieve all the possible combinations of the rectangle and points pairs. ST_Within implemented in phase 1 is used here to determine if they lie within the boundaries of the rectangle. Finally, the results are grouped together and based on their hotness, which is calculated using the containment of the number of points.

5.2.2 Hot Cell Analysis. After performing the above task, we have retrieved the hottest cells. Thus we know its coordinates, and sort operation is performed on them by making use of the Z values. Tests were run on the application using a different configuration of clusters, of which, some of the metrics for the configurations are attached below. We focused on the CPU, network usage and also the network used.

6 RESULTS

An analysis of the memory, CPU and network utilization of the cluster while performing the tasks in the project was conducted. The same functions were run with sample data and the big data-set on three different types of clusters. The metrics for the cluster are collected using Ganglia, and the results are presented. The set of observations were recorded and they are illustrated further below.

6.1 Run1: Small Data, small single node Cluster

The experiment was run with a single node t2.medium instance. The dataset used was the sample datasets for the corresponding problems. Analyzing the results of the experiment, it was observed that the CPU, Memory usage spiked during the execution of the code, often reaching more than 70% load.

6.2 Run2: Large Data, small single node Cluster

The second run of the experiment was with a single node t2.medium cluster, with the full dataset for HotCellAnalysis (2GB of nyctaxitrip data). This experiment caused the spark program to crash due to out of memory errors. It is depicted in Fig 4

6.3 Run3: Large Data, small 3 node Cluster

The worker nodes were started, and hadoop and spark clusters were started as well. The test was repeated, and this time it managed to execute completely. The load was evenly spread out to the workers, thus decreasing the load for the master. This led to decrease in overhead for the master.

6.4 Run4: Large Data, large 1 node Cluster

The Big data test was repeated after upgrading the master to a t2.large instance type. This time, the execution completed, and the master had significant load on the CPU and Memory. The load on the CPU is depicted in Figure 3

6.5 Run5: Large Data, Large 3 node Cluster

The final run of the test was to run the big dataset on a 3 node cluster of t2.large machines. The slaves were also upgraded to t2.large, and the hadoop and spark services were started. The experiement was repeated again, and the network transfer results are graphed in Figure 2. The memory usage of the cluster's master node is also graphed in Figure 5.

From the above multiple runs of the tests, it is clear that the size of the data plays a major role in the load it places on the system. Moreover, the execution time of the tests for the same data, on

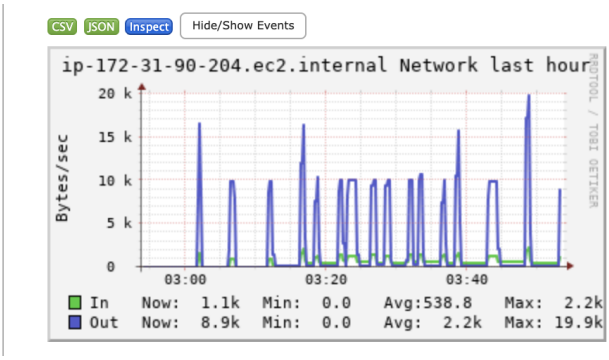

Figure 2: Large-allnodes-Big Data

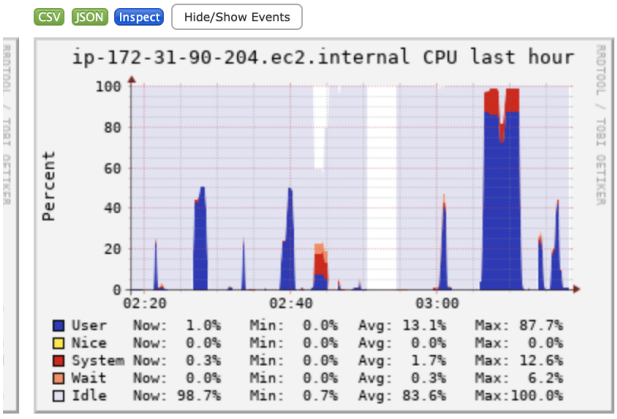

Figure 3: standalone-bigdata

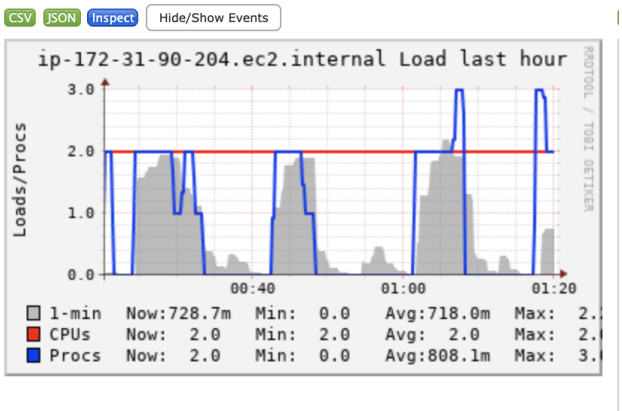

Figure 4: standalone - 2 cores

different clusters were found to be relatively similar compared to the other usage metrics, and hence it is not discussed in detail.

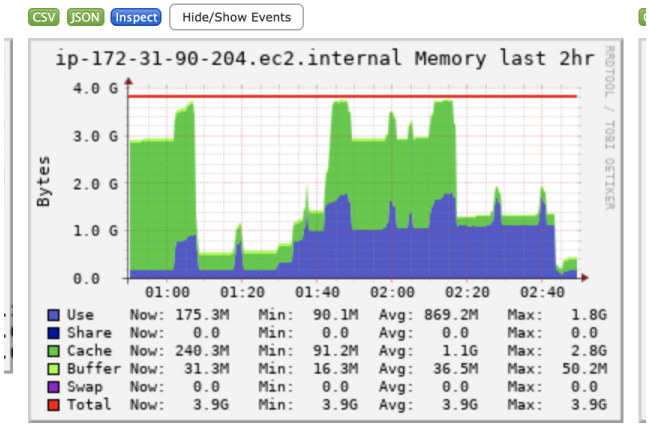

Figure 5: Large-allnodes-Big Data

7 CONCLUSION

The project extensively focused on exploiting the readily available myriad data for the sake of providing infrastructure and services to the public by extracting unusual patterns observed in the dataset. In order to do so, we had created Hadoop and Spark clusters on Amazon Web Service Instances. Geospatial SparkSQL queries like the distance join, space join, and range join were implemented on it and were tested for computing capabilities for the distributed nature of the system. The key takeaways from this project, apart from the extensive knowledge base on distributed systems, were to program in Scala and SparkSQL. The Hadoop, Spark and MapReduce were compared against each other for their capabilities. By monitoring the performance of various combinations of the system architectures, their potential and prowess were well understood over a vast data - the data on the New York City Taxi Trip. We had experimented with the scaling up and scaling out of the architecture and tested their performances.

REFERENCES

[1] 2019. Ganglia in AWS. https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-ganglia.html. Accessed: 2019-04-30.
[2] 2019. Phase 1 Template. https://github.com/YuhanSun/CSE512-Project-Phase1-Template. Accessed: 2019-02-25.
[3] 2019. Phase 2 Template. https://github.com/YuhanSun/CSE512-Project-Phase2-Template. Accessed: 2019-04-07.
[4] Paras Mehta, Christian Windolf, and Agnès Voisard. 2016. Spatio-temporal hotspot computation on Apache Spark (GIS Cup). In *24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*.
[5] J Keith Ord and Arthur Getis. 1995. Local spatial autocorrelation statistics: distributional issues and an application. *Geographical analysis* 27, 4 (1995), 286–306.
[6] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. 2010. The hadoop distributed file system.. In *MSST*, Vol. 10. 1–10.
[7] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2015. Geospark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 70.
[8] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. 2016. Apache spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (2016), 56–65.