A Comparative Analysis of Geo-Spatial Hotspot using Apache Spark

Arpit Kapadia
Arizona State University
Tempe, Arizona
arpit2293@gmail.com

Mahima Gupta
Arizona State University
Tempe, Arizona
mrgupta3@asu.edu

Tejasi Palkar
Arizona State University
Tempe, Arizona
tpalkar@asu.edu

ABSTRACT

The paper tries to disintegrate the performance of Spark clusters while running complex queries through various tests and performance metrics and analyse execution time, network usage and memory utilization over different cluster sizes. Spark is a powerful distributed data preprocessor which is being extended in our project through GeoSpark clustering system over Hadoop Distributed File System(HDFS). Our project comprises of finding to find the top fifty hotspots for New York City Yellow Cab (January 2015) Taxi Records. GeoSpark supports spatial and geometric queries which increases the speed of query processing compared to a simple join query.

KEYWORDS

Apache Spark,performance,large scale data, Hadoop File System, cluster computing, Distributed Database, Big Data

1 INTRODUCTION

The rise of the information age has made the handling, storing and fast execution of large sets of data a necessary requirement for any growing business. Video streaming sites consume and process more than 65PB of data over a distributed network on a daily basis.[1] These video streaming sites also analyze the data and provide recommendations to users based on it. All this requires enormous amounts of data is to be analyzed to produce results in seconds. The main challenges of data system is now data management, data transfer, data control, data discovery and data handling. The distribution of data has freed centralized systems from inhibition of handling large data by making it more salable , available , reliable and fault tolerant. Since the advent of cloud computing and cluster computing distribution of data about multiple nodes has become simpler over the time. Using Map Reduce over a distributed environment is another benefit to further reduce cost in a parallel environment. The map function partitions the problem data in smaller sub-problems which are computed over the network and sent to the reducer for getting the final result.Map-Reduce uses master and worker nodes for storing and redistribution of data.[4] Apache Spark uses in-memory Map Reduce which saves disk reads and writes. In our project, we are using Spark clusters created and managed on Amazon Web Services for a Geo-spatial analysis. We are using New York Yellow Cab January 2015 data set to output top 50 locations that are best for getting customers. Geo spatial analysis refers is the gathering and monitoring of any kind of geographic data which is used in our project through Getis-Ord Gi* statistic.[3]

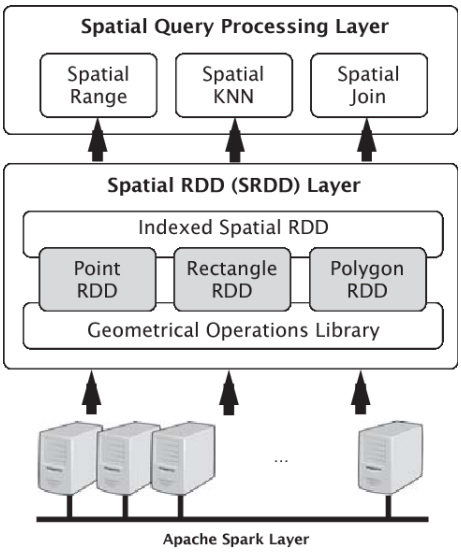

Figure 1: Relation between GeoSpark and Apache Spark[?]

Geo Spark extends Apache Spark and is a computing system used for processing Geo-spatial data. We have taken inspiration from GeoSpark in our project which has proven to perform faster that than a simple join query since it uses Spatial RDD.[2] We created queries to find all the points with a rectangle in Phase 1. In phase 2 we used the simple query to find hot zone locations. Another way proposed in the project was calculating z-score as in [**?**] we came up with top 50 locations for hot cell locations. In this phase, we compare all the spatial queries used for finding best locations based on CPU ,memory and network utilization for each.

2 EXPERIMENTAL SETUP

In Phase one of the project, we set up Hadoop and Spark cluster on the Amazon Web Services EC2 instances. We have a total of 3 EC2 instances running at any given time. All instances are configured so that Password-less SSH to localhost is possible on each of them. Worker instances are set up to be able to bi-direction password-less SSH into the master instance. For phase 2 and phase 3, we have maintained a similar setup. For larger more intensive queries we scale our instances to t2.large to support higher memory requirement.e setup Hadoop 2.7.7 (one Master and two slaves) and Spark 2.3.2 in all the cluster instances. We have utilized auto-scaling of the cloud infrastructure to support larger queries.We created a

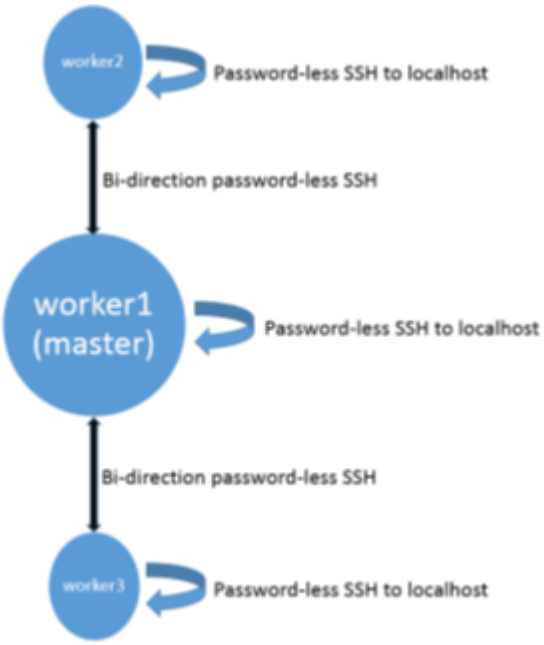

Figure 2: Master Slave Configuration[?]

folder phase1 and phase2 in the home directory and uploaded input datasets and jars in respective folders to support query operations.

2.1 EC2 Details

- Type - t2.micro
- OS-Ubuntu Server 18.04 LTS (HVM) - 64-bit
- Memory - 1GB
- Storage - 8GB
- Hadoop - 2.7.7
- Spark - 2.3.2
- JDK - 1.8.0
- Passwordless bidirectional SSH

2.2 Cluster Setup

We have utilized the Amazon Cloud Watch to monitor our EC2 instances. We have set up the Cloud Watch agent on all of our instances to monitor and return key stats about different metrics and logs. We are using Standard detail level which includes the following metrics to be collected and returned to the console to show visualizations based on them. We have set up different IAM role which allows our CloudWatch agent to collect metrics from the server and optionally to integrate with AWS Systems Manager.

- CPU: cpu_usage_idle, cpu_usage_iowait, cpu_usage_user, cpu_usage_sys
- Disk: disk_used_percent, disk_inodes_free

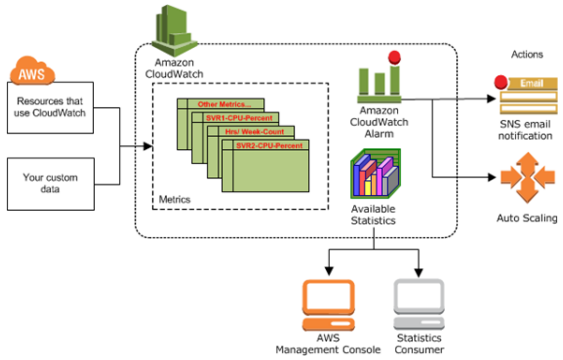

Figure 3: Cloud Watch Management System[?]

- Diskio: diskio_io_time
- Mem: mem_used_percent
- Swap: swap_used_percent

2.3 Commands to Set Up

Create keys for password-less SSH
ssh-keygen -t rsa -P ” -f /.ssh/id_rsa

Reload ssh services with a new configuration
service ssh re

Login to the instance
ssh -i ./instance3.pem
ubuntu@ec2-18-223-24-106.us-east-2.compute.amazonaws.com

Copy files from local system to ec2 instance
scp -ir ./instance3.pem ./phase1
ubuntu@ec2-18-223-126-229.us-east2.compute.amazonaws.com:
/home/ubuntu/Downloads/

Copy files from local system to ec2 instance
scp -i /Downloads/instance3.pem
./CSE512-Project-Phase1-Template-assembly-0.1.0.jar
ubuntu@ec2-3-17-133-163.us-east-2.compute.amazonaws.com:/home/ubuntu/

Submit range query to phase1 jar file created /spark/spark-2.3.2-bin-hadoop2.7/bin/spark-submit
/CSE512-Project-Phase1-Template-assembly-0.1.0.jar
/home/ubuntu/Downloads/output rangequery
/home/ubuntu/Downloads/src/resources/arealm10000.csv
-93.63173,33.0183,-93.359203,33.219456
rangejoinquery /home/ubuntu/Downloads/src/resources/arealm10000.csv
/home/ubuntu/Downloads/src/resources/zcta10000.csv
distancequery /home/ubuntu/Downloads/src/resources/arealm10000.csv
-88.331492,32.324142 1
distancejoinquery /home/ubuntu/Downloads/src/resources/arealm10000.csv
/home/ubuntu/Downloads/src/resources/arealm10000.csv 0.1

3 GEO-SPATIAL QUERIES

For hotspot analysis, we performed the following two tasks:

3.1 HotZone Query

This task requires calculating the hotness of a rectangle based on the number of points located inside a rectangle. A rectangle with more number of points will be considered a hotter rectangle.

3.1.1 Approach. The task involves performing a range join query on rectangle dataset and point dataset. A user defined method called as ST_Contains is used to accomplish the task.The method returns a true if the point coordinate lies within the range of rectangle coordinates and false otherwise.For all the points for which we get ST_Contains as true,a simple range join query is performed between points dataset and rectangle dataset.

3.1.2 Algorithm.

- Load the data from the csv file and create a view
- Get the relevant columns from the data by querying the view in point 1
- Create a view 'point'with data in point 2
- Load data from another CSV that contains the co- ordinates of rectangles and create a view 'rectangle'
- Define and register a function 'ST_Contains'that returns boolean for a given set of rectangle and point coordinated
- Do a join on the 'rectangle'and 'point'tables where ST_contains is True
- Do a groupby(rectangle) and count on the result to get the HotSpots on given data

3.2 HotCell Query

This task applies spatial statistics to spatio-temporal big data in order to identify statistically significant spatial hot spots using Apache Spark. Utilizing spatial statistics to recognize f critical groups or exceptions in spatial information is a surely is a well known logical methodology utilized by GIS experts.
Spatial insights is required when deciding ĂŞ e.g., on the off chance

that we are 95% certain that the dimensions estimated around there surpass an administrative edge, at that point we should react by doing such and such. While distinguishing measurably noteworthy groups (frequently named Hot Spot Analysis), an in all respects generally utilized measurement is the Getis-Ord measurement. It give a z-score and p-values that enable clients to figure out where highlights with either high or low qualities are grouped spatially. Note that this measurement can be connected to both the spatial

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

where x_j = attribute value of cell j,
$w_{i,j}$ = spatial weight between cell i and j,
n = total number of cells,

Figure 4: Gettis-Ord Statistics

and spatio-worldly areas; in this challenge, we are centered around the spatio-fleeting Getis-Ord measurement

3.2.1 Approach. We took in NYC Taxi data and found the top 50 most densely hot cells.The search space is divided into cells The Getis-Ord statistic or Z-Score is used to calculate all the neighbours of a cell, with a total of 26 cells in the cube or search space. Each cube is assumed to be of equal space for convenience. Order the cells according to Depending on the neighbour count of all the cells , the hottest cell is generated.

3.2.2 Algorithm.

- Read data from the csv file, get the columns needed for us, the x,y co-ordinates and the date. Create a view 'nyctaxitrips'from data frame ' pickupinfo '.
- Calculate the boundaries on x,y and z based on the requirements
- Calculate z-score using formula in figure 4
- 'getNeighbour'is a function used to get all the neighbouring cells
- 'checkneighbours'returns true if the specified point is with the bundary otherwise false
- sumofNeighbours = spark.sql("select tc1.x, tc1.y, tc1.z, sum(tc2.count)
  as neighCountReduced, getNeighbours(tc1.x, tc1.y, tc1.z) as neighCount from tripscount tc1 CROSS JOIN tripscount tc2 where checkneighbours(tc1.x, tc1.y, tc1.z, tc2.x, tc2.y, tc2.z) group by tc1.x,tc1.y,tc1.z")
- Create and register a function get zscore that takes neighCountReduced and neighCount from the table query and returns the zScore of the cell.

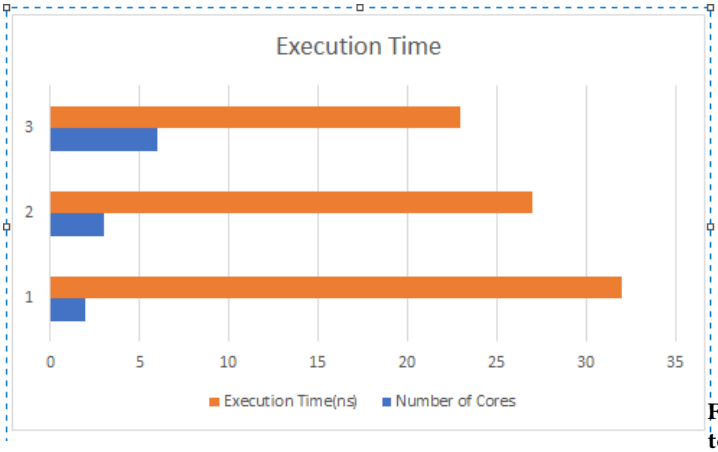

Figure 5: Relation between Execution Time and Cores in Hot Zone Analysis

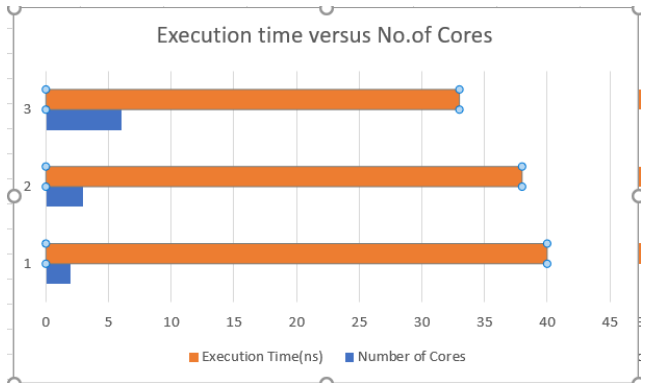

Figure 6: Relation between Execution Time and Cores in Hot Cell Analysis

4 EXPERIMENTAL ANALYSIS

4.1 Execution time

Following inferences were made during execution time analysis:

- The execution time is directly proportional with the number of cores available to perform execution because it was observed during the experiment, that as the number of cores are increased, there was a significant drop in the execution time of HotZone Analysis and HotCell Analysis.
- To understand the impact of cores on processing, experiments were performed for calculating the execution time, we added slaves to the system and comparison of the running time on query execution was made. With increased number of cores, all the tasks were executed faster and the query run time was reduced.

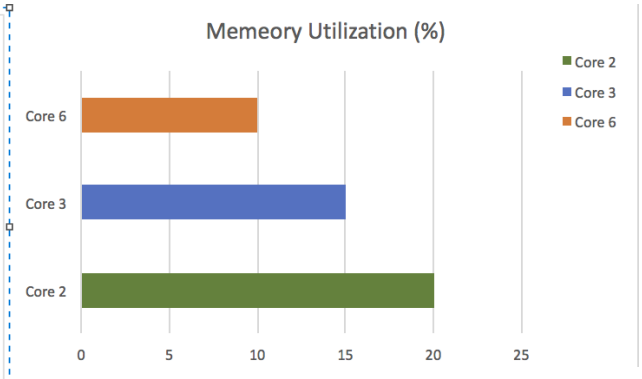

Figure 7: Relation between Memory Utilization and Executors in Hot Zone Analysis

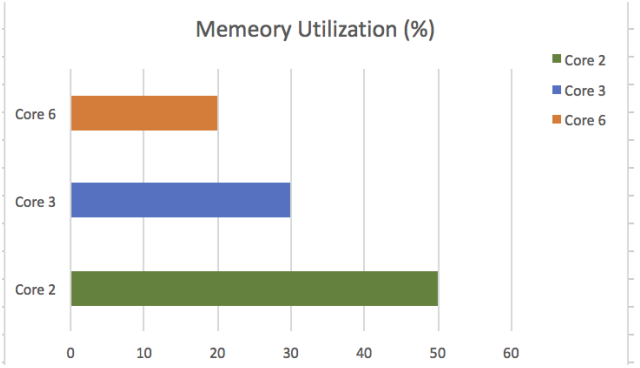

Figure 8: Relation between Memory Utilization and Executors in Hot Cell Analysis

4.2 Memory Utilization

Following inferences were made about memory utilization

- Figure 7 and Figure 8 shows how increasing the cores decreases the memory load
- The load on the all the executors in the cores is significantly reduced which leads to reduced memory utilization

4.3 Throughput

Following inferences were made about system throughput when the number of cores were increased:

- Throughput is defined as the rate at which data is transferred over a network .As in Figure 9. and Figure 10. we can see the the throughput remains the same,irrespective of the number of cores.It can be attributed to the fact that throughput is measured as the ratio of network_in and network_out, the value of which remains more or less the same.

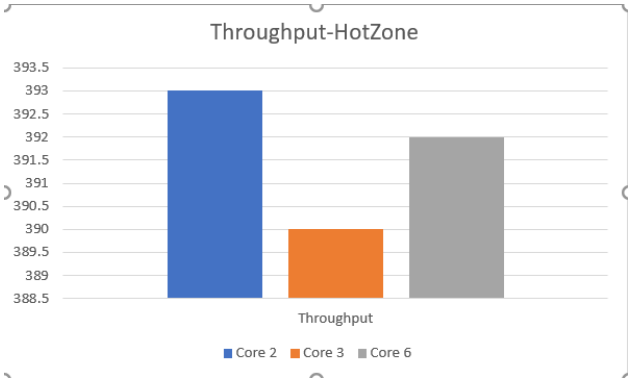

Figure 9: Throughput Analysis:HotZone

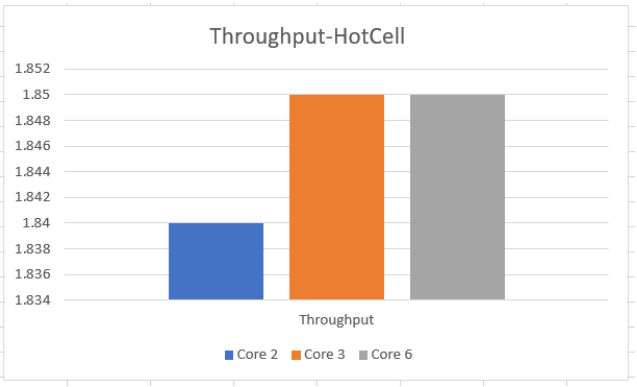

Figure 10: Throughput Analysis:HotCell

5 CONCLUSION

The project was an attempt to understand Spark and Hadoop Distributed File System. We have tried to evaluate Spark cluster and analyzed the execution time, throughput and memory utilization of the system when we perform HotZone and HotCell Analysis in our project .For this phase of the project, we devised and ran a number of tests which helped us to analyse system performance when the no.of cores executing the task were increased. We have studied and reported the results for execution time, throughput and memory utilization through our analysis for the above two methods.

6 ACKNOWLEDGEMENTS

We would like to thank Dr.Mohamed Sarwat and TA Yuhan Sun for their continuous guidance and recommendations to carry out the analysis task.

REFERENCES

[1] Databricks. [n. d.]. *Netflix's Recommendation ML Pipeline Using Apache Spark - Databricks LaTeX.* https://databricks.com/session/netflixs-recommendation-ml-pipeline-using-apache-spark.
[2] Mix and Match:Agile Flavors . [n. d.]. *https://www.linkedin.com/pulse/mix-match-agile-flavours-samir- mishra-prince2-csm-safe-agilist/.*
[3] Agilenutshell.com. (Extreme Programming. [n. d.]. *http://www.agilenutshell.com/xp.*
[4] Wikipedia. [n. d.]. .