Apache Spark performance Evaluation using Spatial Queries

Ganesh Kumar Betha
gbetha@asu.edu
CIDSE, Arizona State University
Tempe, Arizona

Akshay Kumar Gunda
agunda1@asu.edu
CIDSE, Arizona State University
Tempe, Arizona

Sai Srujan Jaligama
sjaliga1@asu.edu
CIDSE, Arizona State University
Tempe, Arizona

ABSTRACT

This paper evaluates the performance of a spark distributed systems. Spatial queries on Geometry databases. Various parameters like number of data nodes in each cluster and the memory size are varied and the performance of the system in terms of run time, memory usage, CPU utilization, communication cost, and network data usage are evaluated. The results are reported and discussed for various possibilities.

KEYWORDS

Apache Spark[1], Hadoop[2], Scala, Virtual Machines, Hadoop File System, Google Cloud Platform, Amazon EC2, Range Query, Range Join , Distance Query, Distance Join , Hotzone Analysis, Hotcell Analysis.

1 INTRODUCTION

There has been an increase of data in the last decade due to affordable hardware and penetration of Internet in newer geographical areas. Analyzing the ever increasing data in a time efficient manner has become a problem in recent days for any organization. The demand to store and process large data gave rise to new parallel/distributed computing frameworks like Hadoop and Spark which tries to minimize the computation time by distributing the work among different machines.

Apache Hadoop is a collection of software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. The core of Hadoop consists Hadoop Distributed File System (HDFS)(storage part), MapReduce programming model(Programming part). Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data in parallel. Hadoop takes advantage of data locality, where nodes manipulate the data they have access to. This allows Hadoop to process faster and efficient than conventional super computers.

Apache Spark is a cluster-computing framework which provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Spark was developed in response to the limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs. Spark facilitates the implementation of both iterative algorithms, which visit their data set multiple times in a loop, and interactive/exploratory data analysis(or repeated database-style querying of data).

The purpose of the project is to evaluate performance of the spatial query methods while changing the configuration of Spark clusters. The dataset used to run this experiment is the spatial geometry database[3]. The details of the query methods and dataset used to evaluate the performance are discussed in the following sections. The metrics and the results obtained are also explained in further sections.

2 METHODS AND DATASETS USED

To evaluate the performance of the distributed system, The queries (and corresponding methods) developed in the phase-1 and phase-2 of the project are used. The queries used in the experiment are spatial queries. A spatial query is a special type of query supported by **Geo-databases** and **spatial databases**. Spatial queries differ from traditional SQL queries in that they allow for the use of points, lines, and polygons. These queries also consider the relationship between these geometries.

The details of the spatial queries and the datasets used are described in the next section.

2.1 Methods

The following spatial queries, which were implemented in phase-1 and phase-2 of the project are used to evaluate the performance of the distributed system.

- **Range query:** Given a query rectangle R and a set of points P, Range query finds all the points within R. This query utilizes ST_Contains function.
- **Range Join Query:** Given a set of Rectangles R and a set of Points S, Range join query finds all (Point, Rectangle) pairs such that the point is within R. This query utilizes ST_Contains function.
- **Distance query:** Given a point location P and distance D in km, Distance Join Query finds all the points that lie within a distance D from P. This Query utilizes ST_Within function.
- **Distance Join Query:** Given a set of Points S1 and a set of Points S2 and a distance D in km, Distance Join Query finds all the pairs of the kind (s1,s2), such that s1 is within a distance D from s2. This Query utilizes ST_Within function.
- **Hotzone Analysis:** Hotzone Analysis utilizes ST_Contains to find the hotness of a rectangle, i.e, the number of points contained by it.
- **Hotcell Analysis:** Hotcell Analysis deals with applying spatial statistics to spatio-temporal big data in order to identify statistically significant spatial hotspots. This is done by calculating Getis-Ord Statistic[3] for each cell.

2.2 Dataset

The following datasets are used to evaluate the performance of the system:

- **arealm10000.csv:** Used by Range, Range Join, Distance and Distance Join Queries. It contains 10,000 points. It is about 217KB in size.
- **zcta10000.csv:** Used by Range Join Query. It contains 10,000 rectangles and is about 420KB.
- **zone-hotzone.csv:** Used by Hotzone Analysis. It contains 286 rectangles. It is about 13KB.
- **point-hotzone.csv:** Used by Hotzone Analysis. It contains 10,000 New York City taxi trips with pickup coordinates and is about 1,811 KB.
- **yellow_trip_sample_100000.csv:** Used by Hotcell Analysis. It contains data of 100000 New York City taxi trips. It is about 18MB.

3 EXPERIMENTAL SETUP

3.1 Cluster Setup

Initially We used Amazon EC2 instances to perform the experiments, but as the AWS provides only 1 GB RAM with 1 core for free tier we were unable to perform experiments by changing the configuration of the cluster, as the only option to change the configuration was to decrease the RAM size to 0.5 GB, using 0.5 GB RAM caused the system to crash as the dataset is huge. This made us change to Google cloud Platform.

We created a cluster setup on Google cloud Platform's *Data Proc service* [4] with Hadoop(version 2.7.7) and Apache Spark(version 2.0.2) to run our experiments. The *Data Proc service* [4] provides easy manipulation of cluster configuration(for example, change of the memory allocated, changing the number of cores and the changing the number of data nodes). In google cloud, we used VM instances with 7.5 GB RAM(limited the memory of spark executors to maximum of 6 GB) in two core instances.

3.2 Evaluation metrics

The evaluation metrics are described as below.

3.2.1 ***Execution Time:*** It is the total time taken by the cluster to execute a Spatial Query. This metric is measured in seconds i.e. the number of seconds a query executes.

3.2.2 ***CPU Utilization:*** It determines the percentage of work handled by a CPU to execute a Spatial Query. This metric is measured in percentage(%) i.e. the percentage of CPU utilised.

3.2.3 ***Disk Write Bytes (Memory Utilisation):*** It describes the total memory used by each cluster node to execute a Spatial Query. This metric is measured in Kilo Bytes written per sec (KB/sec) i.e. the total number of Bytes written onto the disk per second by each cluster node.

3.2.4 ***Network Bytes In :*** It describes the total amount of bytes received by each cluster node while executing a Spatial Query. This metric is measured in Kilo Bytes per sec (KB/sec) i.e. the total number of Bytes received by the cluster node per second.

3.2.5 ***Network Bytes Out :*** It describes the total amount of bytes sent by each cluster node while executing a Spatial Query. This metric is measured in Kilo Bytes per sec (KB/sec) i.e. the total number of Bytes sent by the cluster node per second.

4 EXPERIMENTAL EVALUATION

After selecting the Google cloud platform for our distributed environment, we compared the performance by varying the cluster configurations such as varying the **number of data nodes**, **executory-memory (RAM)** which is the memory allocated for each executor in each instance. We observed that these parameters significantly affected the performance of our distributed system. Google Cloud platform allows us to easily increase and decrease the memory for each instance by using following query parameters which allowed us to change the system configuration *–executor-memory*.

For the purpose of our experiments we chose to configure memory allocated using Spark configuration option *–executor-memory*. To change the number of nodes, we started and stopped the instances in google cloud console. Every time we run the script, we captured the *memory utilization*, *average CPU utilization*, *number of bytes transferred across network* from Stack driver monitoring system[5]. The running time was calculated by using time module of python to capture the start time and end time of each submitted job.

We ran our experiment with the below system configurations.

- 3 Machines ,2GB RAM, 2 core
- 3 Machines ,4GB RAM, 2 core
- 3 Machines ,5GB RAM, 2 core
- 4 Machines ,2GB RAM, 2 core
- 4 Machines ,4GB RAM, 2 core
- 4 Machines ,5GB RAM, 2 core

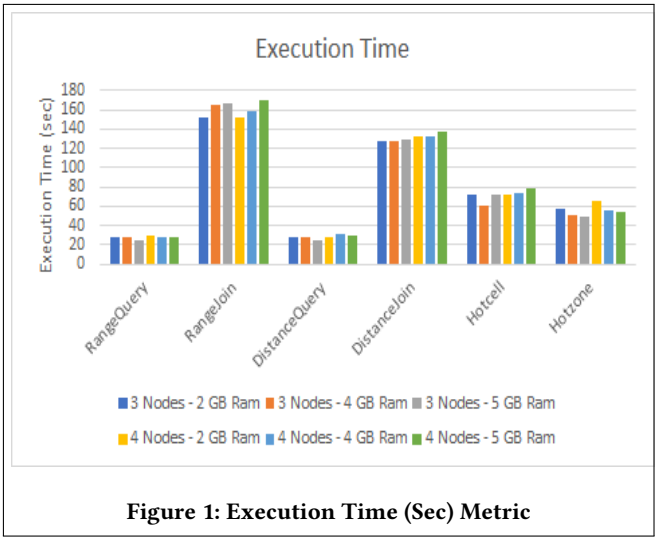

Figure 1: Execution Time (Sec) Metric

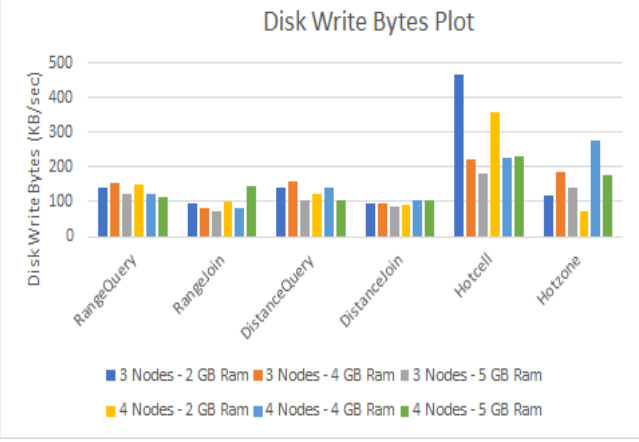

Figure 2: Memory Utilisation (KB/s) Metric

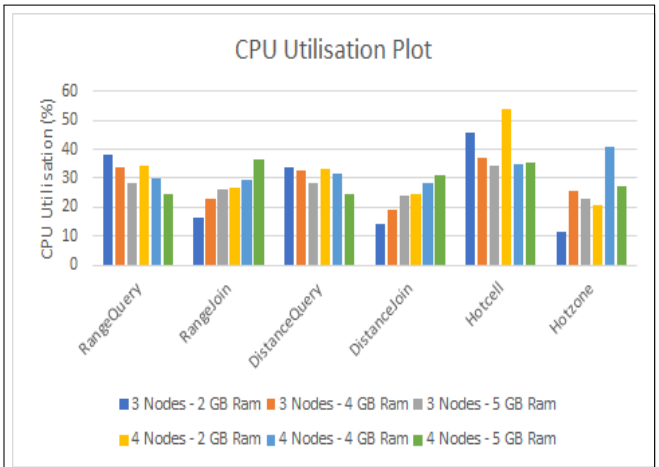

Figure 3: CPU Utilisation (%) Metric

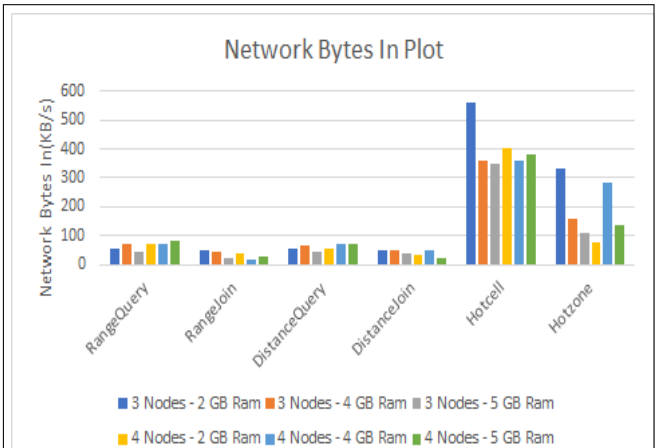

Figure 4: Communication Cost KB/sec) Metric: Network Bytes In

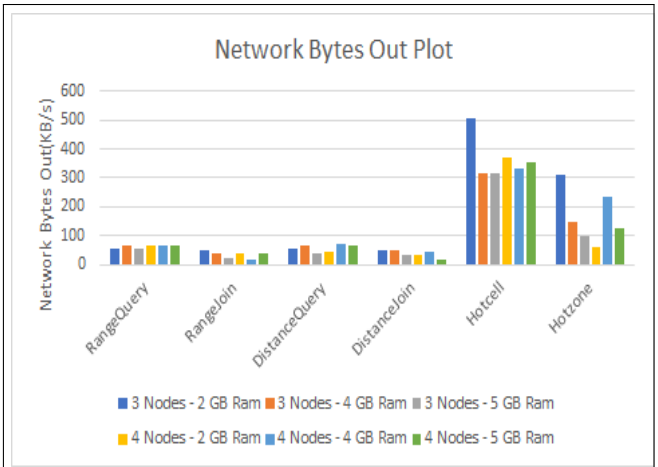

Figure 5: Communication Cost (KB/sec) Metric: Network Bytes Out

4.1 Explanation

Table 1 compares the execution time of various queries on different machine configurations. Execution time of the queries can help us determine the performance of the configuration better. *Hotcell Analysis* is the best query to compare various configurations. As it is a computation intense query, it is sensitive to configuration changes. Execution time decreases with increase in number of nodes in the cluster. As the number of nodes increases, each node has to work on small partition of data as the data is further partitioned into smaller data, this decreases the execution time. As we can see, the change in memory allocated alone does not effect the execution time, as increasing the memory allocated to the nodes, by keeping the number of nodes constant,has no effect in load to individual nodes.

Table 2 compares the memory utilization of various queries on different machine configurations.There is no significant change in memory utilization with change in number of nodes as the spark job will use as much memory as needed. Looking at the metric data, there is no change in memory usage, until large memory is allocated to the machines. In some instances, the memory used exceeded the total allocated memory, which is due to changing the memory parameter using spark configuration. As the VM is configured with 7.5 GB memory, the memory used may exceed the spark allocated memory as per the submitted job.

Table 3 compares the CPU utilization of various queries on different node configurations. The CPU utilization decreases with increase in number of nodes keeping memory allocated constant. This is because, as number of nodes increases, the task is divided

Table 1: Execution Time

Nodes and RAM	Range Query	Range Join	Distance Query	Distance Join	Hotcell	Hotzone
3 Nodes-2GB Ram	28.1	152.65	27.58	127.86	72.52	57.27
3 Nodes-4GB Ram	27.42	164.51	28.49	128.25	61.01	50.81
3 Nodes-5GB Ram	24.57	167.64	25.28	128.69	72.33	49.61
4 Nodes-2GB Ram	29.09	151.62	27.79	132.36	71.42	65.24
4 Nodes-4GB Ram	28.44	157.91	30.38	131.95	73.34	55.36
4 Nodes-5GB Ram	27.52	170.18	29.26	138.18	77.94	53.28

Table 2: CPU Utilization

Nodes and RAM Size	RangeQuery	RangeJoin	DistanceQuery	DistanceJoin	Hotcell	Hotzone
3 Nodes - 2 GB Ram	37.93	16.44	33.89	14.4	45.82	11.37
3 Nodes - 4 GB Ram	33.84	23.1	32.46	18.95	37.13	25.6
3 Nodes - 5 GB Ram	28.37	26.24	28.33	23.95	34.11	22.86
4 Nodes - 2 GB Ram	34.4	26.8	33.21	24.27	54.18	20.6
4 Nodes - 4 GB Ram	30.14	29.47	31.49	28.24	34.87	40.94
4 Nodes - 5 GB Ram	24.6	36.52	24.49	30.93	35.44	27.3

Table 3: Disk Write/memory Utilization

Nodes and RAM Size	RangeQuery	RangeJoin	DistanceQuery	DistanceJoin	Hotcell	Hotzone
3 Nodes - 2 GB Ram	139.2	94.6	141.5	94.3	469.4	117.2
3 Nodes - 4 GB Ram	154.7	79.6	159.5	95.3	224.2	188.3
3 Nodes - 5 GB Ram	122.8	72.8	106.4	88.1	181.9	142.7
4 Nodes - 2 GB Ram	150	100	124	93	358	74
4 Nodes - 4 GB Ram	123	81	139	103	229	276
4 Nodes - 5 GB Ram	113	147	106	106	230	176

Table 4: Network Utilization-In Bytes

Nodes and RAM Size	RangeQuery	RangeJoin	DistanceQuery	DistanceJoin	Hotcell	Hotzone
3 Nodes - 2 GB Ram	55.66	51.24	55.82	50.89	562.6	330.9
3 Nodes - 4 GB Ram	70.07	44.09	63.45	51.48	359	157.4
3 Nodes - 5 GB Ram	43.6	21.33	41.97	36.25	350.6	107.6
4 Nodes - 2 GB Ram	73	38	52	35	403	76
4 Nodes - 4 GB Ram	73	17	73	47	362	283
4 Nodes - 5 GB Ram	82	29	71	20	383	134

Table 5: Network Utilization-Out Bytes

Nodes and RAM Size	RangeQuery	RangeJoin	DistanceQuery	DistanceJoin	Hotcell	Hotzone
3 Nodes - 2 GB Ram	52.01	48.03	51.86	46.83	509	311.09
3 Nodes - 4 GB Ram	64.06	35.6	64.86	48.22	314.3	147.5
3 Nodes - 5 GB Ram	53.28	20.78	38.17	35.3	314.7	100.63
4 Nodes - 2 GB Ram	67	36	46	32	371	58
4 Nodes - 4 GB Ram	68	15	69	45	335	236
4 Nodes - 5 GB Ram	64	38	67	17	355	124

on more number of nodes which decreases the CPU utilization in individual nodes. However, it may not necessarily decrease with increase in memory, as increasing the memory allocated to the nodes alone,has no effect in load to individual nodes.

Table 4 and *table 5* compares the communication cost(in-bytes and out-bytes) of various queries on different machine configurations.The Communication cost is effected with change in number of nodes. As the number of nodes increases, computation is distributed among the nodes which increases number of bytes to be transfer among the nodes. This increases Network Bytes in and Network Bytes out.

5 CONCLUSION

We have measured the performance of the spark system by varying the number of nodes in a cluster and also varying the RAM size. We used spatial queries (of varying complexities) and geometry dataset to perform this experiment. The number of machines we used for the experiment is 3 and 4. The RAM sizes of 2GB, 4GB and 5GB was used. The results suggest that as the number of machines in the cluster is increased, there is a decrease in the run-time. There is also decrease in memory utilization as the number of machines increased from 3 to 4.

6 REFERENCES

[1] Apache Spark, https://spark.apache.org/
[2] Apache Hadoop, https://hadoop.apache.org/
[3] Ord, J.K. and A. Getis. 1995. "Local Spatial Autocorrelation Statistics: Distributional Issues and an Application" in Geographical Analysis 27(4).
[4] Google Cloud Platform, Data Proc Service, https://cloud.google.com/dataproc
[5] Google Cloud Platform, Monitoring Service, https://cloud.google.com/monitoring