Performance Analysis of GeoSpatial Queries

Keshin Jani
Arizona State University
khjani@asu.edu

Swarnim Sinha
Arizona State University
ssinha39@asu.edu

Upinderjit Singh
Arizona State University
usingh17@asu.edu

ABSTRACT

The goal of this project is to experimentally analyze the run time performance on a single node spark cluster and multi-node spark cluster using various performance indicators like execution time, CPU utilization and memory utilization. This testing was done using the open-source tool Ganglia to understand the behaviour of the above-mentioned metrics for a user-defined script for geospatial data.

1. INTRODUCTION

The objective of this paper is to give a detailed explanation of the performance of an application implementation on Apache Spark. The various performance metrics analyzed include the execution time, CPU utilization and memory utilization. The application performs various routines such as ST-Contains, ST-Within, Hot Zone Analysis, and Hot Cell Analysis. These routines were tested on both single and distributed Apache Spark systems.

Apache Spark is an open-source distributed cluster computing framework based on Resilient Distributed Dataset[1]. This makes Apache Spark a fault-tolerant system which supports both stream processing and data parallelism. Fault tolerance is accomplished by maintaining a history of transformations implemented to the data objects.

Spark applications are executed as separate processes on a cluster as in a master-slave architecture as shown in Fig 1. The SparkContext is the master that communicates to the nodes in the clusters which are the slaves through the cluster manager. These worker nodes or the executors implement the jobs allocated to them and die as soon as the implementation of application is over. For our analysis in this experiment we modify various aspects included in the environmental parameters like memory, storage and number of cores for the node. Ganglia which is a distributed cluster monitoring tool is used for the evaluation of various performance metrics of the application being implemented on the Apache Spark cluster such as execution time, memory usage and CPU utilization.

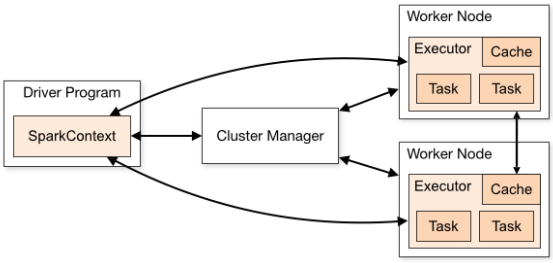

Fig 1. Spark Architecture[1]

2. EXPERIMENTAL SETUP

For the experimental analysis two types of Spark clusters namely single node and multi-node cluster. Hadoop-Yarn was used as the platform to provide a framework for job scheduling and cluster management of resources. Yarn executes an independent master-slave implementation when the Spark application is implemented on top of it. Yarn permits the use of Yarn command line operations such as:

- Executor-cores - number of cores required by an executor

- Driver-memory - CPU utilization of the driver
- Num-executors - number of executors
- Executor-memory - memory allocation for an executor

Another advantage of Yarn is its ability of dynamic sharing and central configuration of the set cluster resources amongst all the frameworks running on top of it.

2.1 MACHINE CONFIGURATION

In this set up we used 3 Elastic Compute Instances of Amazon Web Services in a master-slave architecture with one master and two slave instances. Amazon Web Services is a leading cloud computing platform which provides cloud instances on the basis of demand[2]. We set up the communication between the master and slave with no firewalls and in the same private cloud environment. With no firewalls rules on the instances, all kinds of communication can flow from one machine to another machine freely. We also added the machine's public key to another machine's authorized key set in this way we activated the password-less SSH. We used machines with specifications 16 GB RAM and vCPU 4 per machine.

2.2 PARAMETER TUNING

In this experimental setup, we have different parameters like number of executors, memory allocation and computation power. With multiple permutations possible for the given parameters we set a basic parameter configuration which we increased after one round of performance checks.

We used Ganglia, an open-source monitoring system, to record and check the performance of the task on the given system with set parameters. This system provides CPU and memory utilisation for the given system on a real-time basis. We run the experiments with different parameters which can be set using two approaches namely static and dynamic. Static approach is the one in which we manually change parameters in the spark-submit command. A dynamic approach is the one in which we increase or set the parameters using the spark-session variable. We performed experiments to check the system's performance for Hot Zone analysis, Hot Cell analysis, ST within and ST contains scripts.

In this experiment, we set the number of executors and cores both from 1 to 3 and input data from 1-2GB. While varying the parameters with set guidelines we measured and visualized the system performance. We were measuring performance under different categories one of them was run time for the application and also CPU consumption.

We measured the utilization of the CPU using Ganglia[3] and the graphs it generated. Ganglia consists of 3 main parts - Monitoring thread, Meta Thread and Front end[4]. The Ganglia Monitoring thread is used to store the data in-memory and pay attention to the messaging bus. It then replies to every request with an XML output for the information of the cluster state. Master nodes are run by the Master thread and it also gathers the information from different machines. The front end part is where the graphs and visualisation of the data takes place. The parameters of the spark job are then set dynamically which allows the number of executors and memory to be set for the spark job.

3. RESULT EVALUATION

We have divided this experiment in three parts:

- Varying number of cores on a single machine.

- Varying amount of input data.
- Varying number of nodes in a cluster.

Each above section will be tested (and visualized in the same format) on the basis of below-mentioned division of the Project.

- Project Phase-1 Range query and Range Join

In the first phase of the project, we had the program a ST_Contains function in the given template for range query and range join function. In this ST_Contains function for range query, given a set of points P and a query rectangle R we find all the points in the rectangle R. For the range join query, we are given a set of rectangles and points, we find all the pairs of rectangles and points which lie within the rectangle.

- Project Phase-1 Distance query and Distance Join

We implemented ST_Within program in the given template for distance join and distance query function. In the distance query with ST_within function, given an input of a location P and distance D, we find all points which are within a distance D from the point P. In the distance join query with ST_within function, given a set of points S1, S2 and distance D, we find all pairs of points (S1, S2) which are a distance D apart.

- Project Phase-2 Hot Zone and Hot Cell Analysis

In hot zone analysis, we perform range join on datasets of rectangle and point. We find the count of points within a rectangle and this defines the hotness of the rectangle. In Hot cell analysis, we need to calculate the G-score for every cell. This score defines the hotness of the cell. We then descendingly order the cells according to the G-score.

3.1 NUMBER OF CORES

First set of evaluation plans includes changing the number of cores used on one machine.

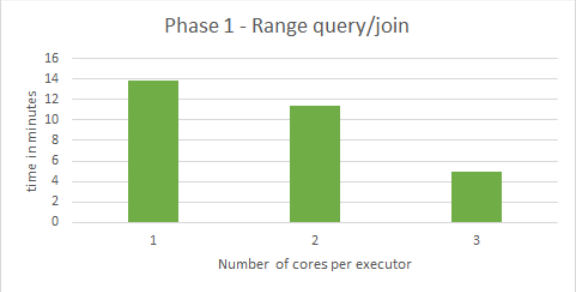

Fig-2: Cores v/s Runtime analysis for range query/join

Fig-3: Cores v/s Runtime analysis for distance query/join

Fig-4: Cores v/s Runtime analysis for HotZone & HotCell

As we can see from the above results (Fig-2,3,4) that increasing the number of cores on one machine reduces run-time execution for each project phase. The reason being the parallel

execution among multiple cores of a machine. I.e. A single processor runs instructions on different cores at the same time.

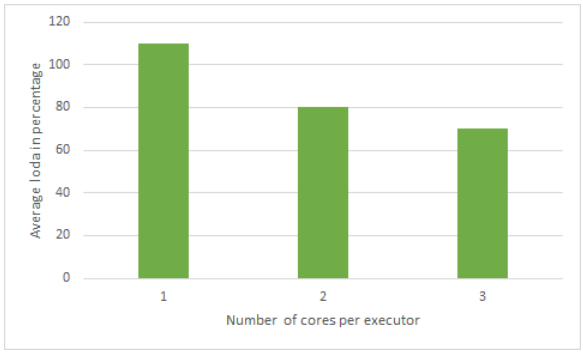

Fig-5: Load distribution

The above graph (Fig-5) depicts an average CPU consumption to run the Project phases with different numbers of core per machine.

3.2 SIZE OF INPUT DATA

For this part of the experiment we have used the input data with two different sizes: 1 GB and 2GB. This step was done manually by providing different sized files as an input to the three commands.

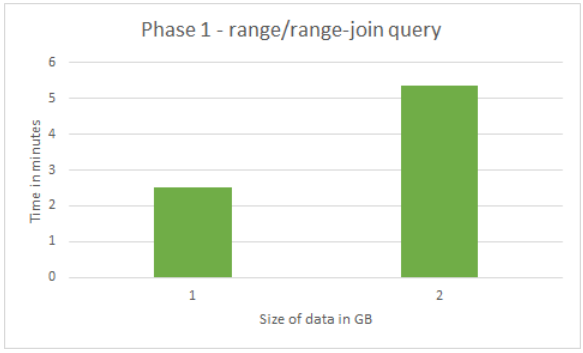

Fig-6: I/p Size vs Runtime analysis for range query/join

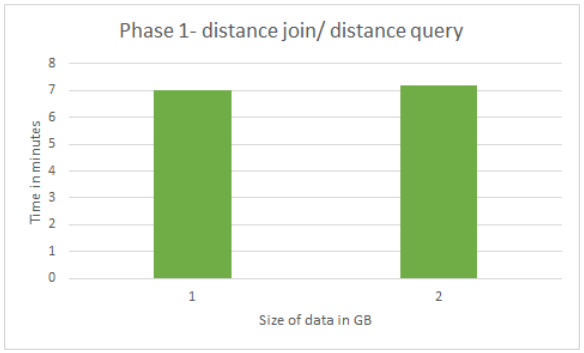

Fig-7: I/p Size vs Runtime analysis for range query/join

Fig-8: I/p Size vs Runtime analysis for HotZone & HotCell

Fig-6,7,8 shows the expected results. Although the execution time for 2 GB input data is more than execution time for 1 GB data for each setup, the difference for the distance query/join and hotcell/hotzone (Fig-7,8) setup was comparable.

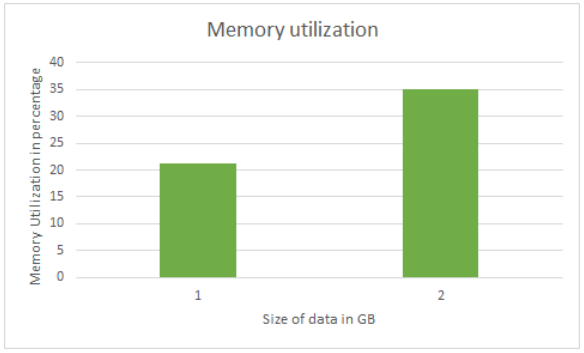

Fig-9: I/p Size vs Memory utilization in percentage

We also compared memory utilization (Fig-9) while varying the input data sizes. The percentage of memory utilized for 1 GB data was 21.3% and for 2GB it rose up to 35% as depicted below.

3.3 NUMBER OF NODES

For this part of the experiment, we are varying the number of nodes in a distributed cluster. To perform this we used Amazon EC2 setup as was set up in Project Phase-1.

- Input data = 2GB
- Number of cores per executor = 1

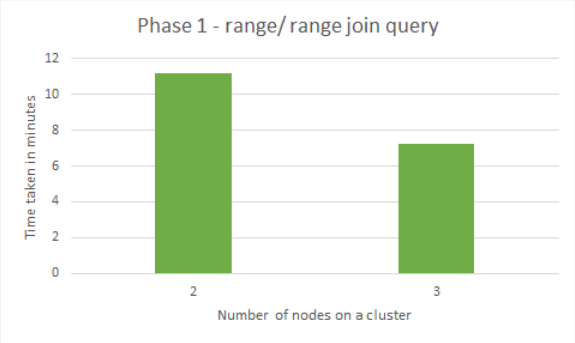

Fig-10: Nodes v/s Runtime analysis for range query/join

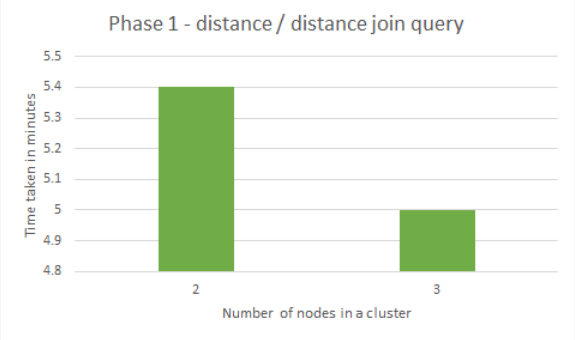

Fig-11: Nodes v/s Runtime analysis for distance query/join

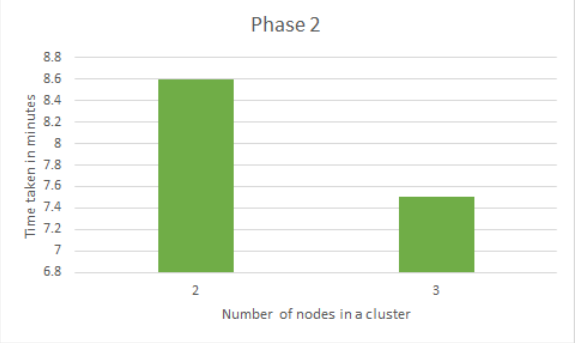

Fig-12: Nodes v/s Runtime analysis for HotZone & HotCell

We are depicting results for 2 and 3 nodes only because for 1 node the results are already shown in section 4.2

As we can infer from the results (Fig-10,11,12) that the execution time for each Project phase is inversely proportional to the number of nodes used in the cluster, i.e., more the number of nodes more is the performance efficiency of the cluster.

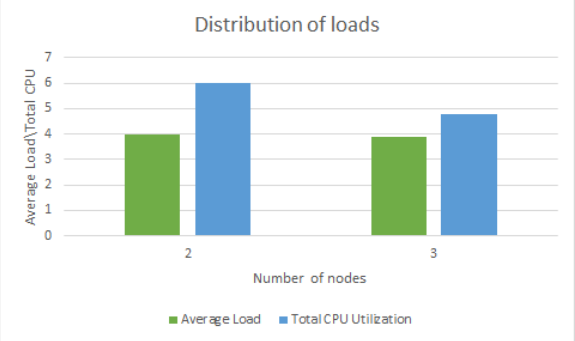

Fig-13: Load Distribution & CPU utilisation

From the above graph (Fig-13), we could see that the utilization of CPU was higher in a cluster with 2 nodes, as compared to 3 nodes. We also noted that the average load per executor was lower when the number of nodes increased from 2 to 3.

4. CONCLUSION

We successfully performed experiments in a distributed system using spark for phase one and phase two projects. We varied different systems parameters while executing the experiment and presented a detailed analysis for the same through our analysis and visually through graphs as well. It gave us hands-on experience of working in a distributed system for writing code and executing binaries for the same.

5. ACKNOWLEDGEMENT

We, as the authors of this report, would like to thank Professor and TA for their constant guidance and support throughout the course which helped us understand the core distributed database concepts and successfully write this analysis paper.

6. REFERENCES

[1]https://spark.apache.org/docs/latest/cluster-overview.html

[2]https://aws.amazon.com/what-is-aws/?nc1=f_cc

[3]https://www.oreilly.com/library/view/data-analytics-with/9781491913734/ch04.html

[4]http://www.agildata.com/apache-spark-cluster-managers-yarn-mesos-or-standalone/