Distributed and Parallel databases - Project Phase II

I. Introduction

This project phase aims at comparing the various Spatial join, range and nearest neighbor operations on GeoSpark running on a Hadoop cluster, by monitoring cluster resources like memory usage, CPU usage and execution time, averaged over a few runs.

II. Cluster setup

We setup 3 Ubuntu machines running Hadoop 2.6.4 and Spark 2.0.1.

System	Memory	CPU	Cores
Master	2.9 GB	Intel i5	2
Worker1	6.7 GB	Intel i5	2
Worker2	6.7 GB	Intel i7	4

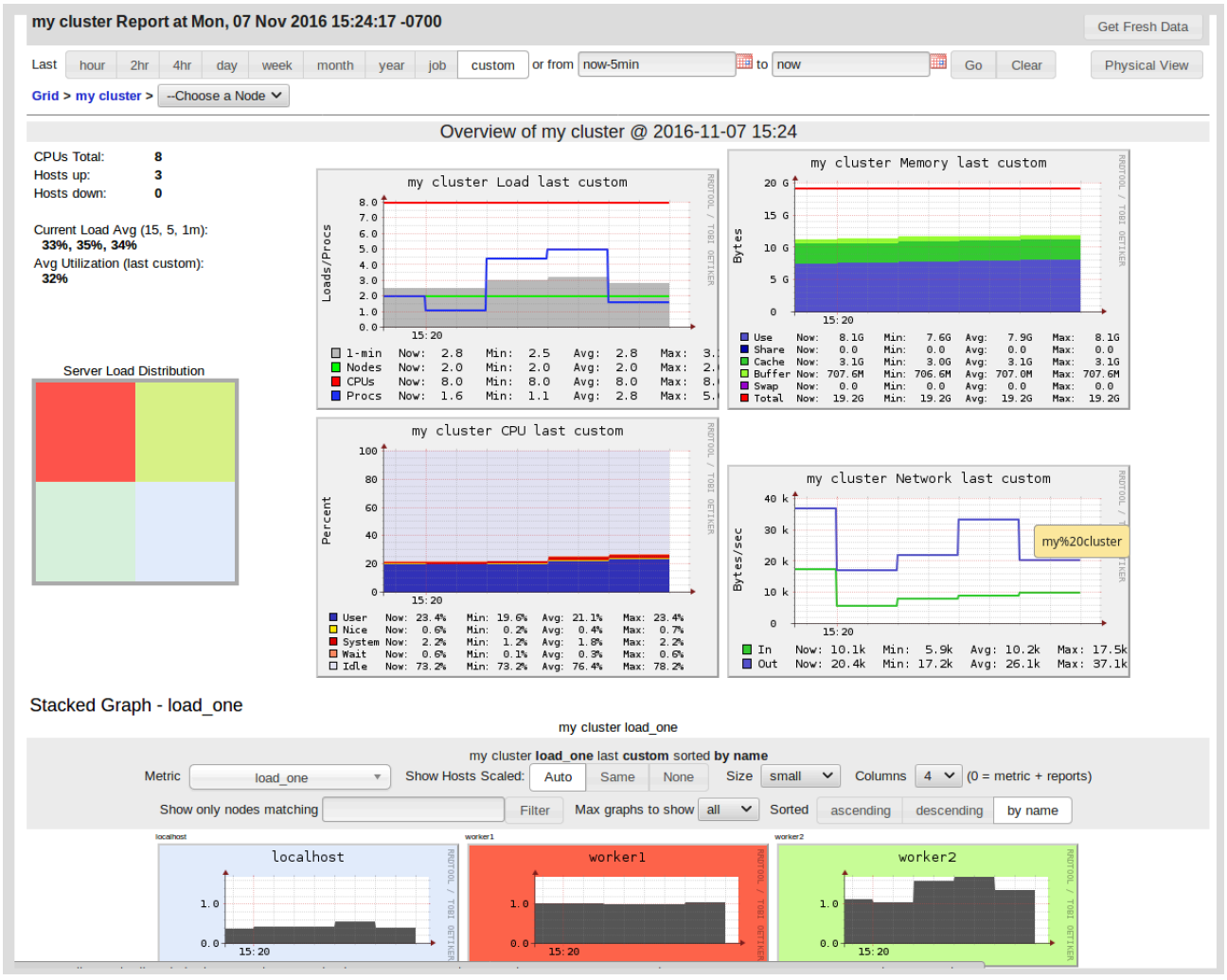
Collecting metrics

In order to collect metrics, we installed ganglia daemon on the master node and ganglia monitoring on the worker nodes.

From the ganglia UI, we created a window in the monitor dashboard that would capture each query's execution. For e.g. the queries 2a, 2b, 3a, 3b which run within a minute, we set a 1-minute window and collected the average value for the cluster's memory and CPU utilization. Similarly for 4a, 4b, 4c we set a 5-min window and for the phase 2 query we set a 3 hr window. We run the queries multiple times and set window sizes accordingly and collect averages over the window.

For execution time, we used the age-old method of collecting start and end time in the scala shell and taking their difference to yield the query execution time.

```
scala> val task4c_st=System.currentTimeMillis();
val result3 =
joinQuery3.SpatialJoinQuery(pointRDDWGridRTree,rectangleRDD).count;
val task4c_et=System.currentTimeMillis();
println(task4c_et – task4c_st)
```



III. Queries comparison and explanations

With the above setup, we ran the queries from Project phase I and collected the metrics below:

Comparison of 2a and 2b:

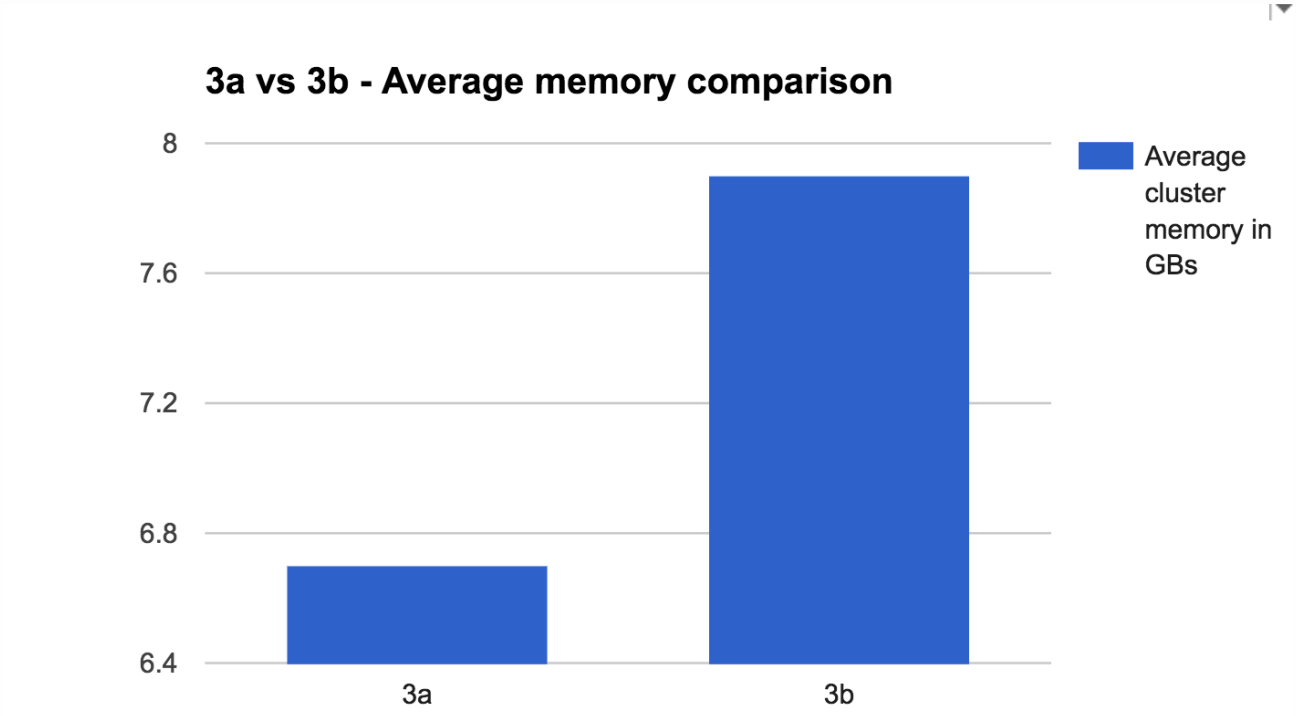
Phase 1	Average Execution time (in milliseconds)	Average cluster memory in GB	Average CPU usage %
2b	1284.333333	9.1	17.26666667
2a	2756.333333	9.2	17.4

Here we see that the indexed range query takes less time than the non-indexed one, which is as expected.

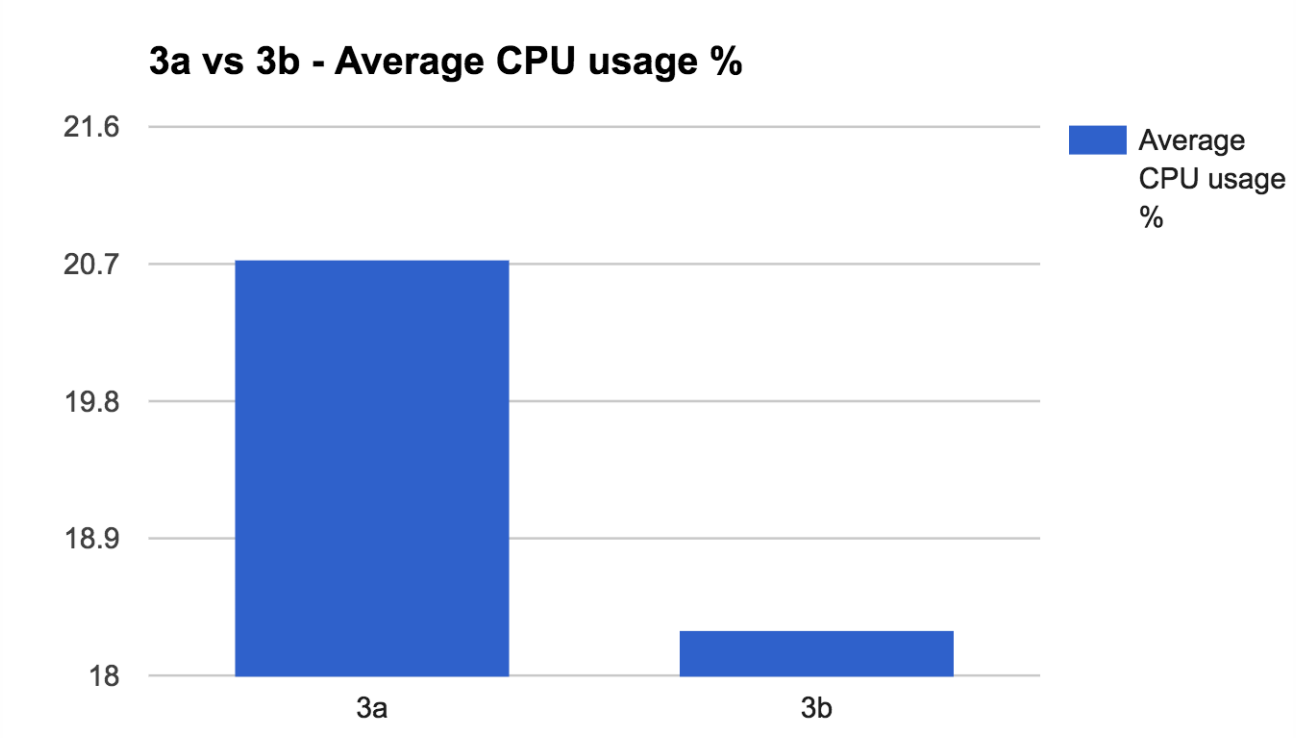
Comparision of 3a and 3b:

Phase 1	**Average Execution time (in milliseconds)**	**Average cluster memory in GB**	**Average CPU usage** *%*
3a	4411.666667	6.7	20.7333333
3b	1778.666667	7.9	18.3

Since the index is precomputed in 3b, the Knn query will have lesser number of shuffle stages than the 3a query. This is why 3b is faster than 3a.

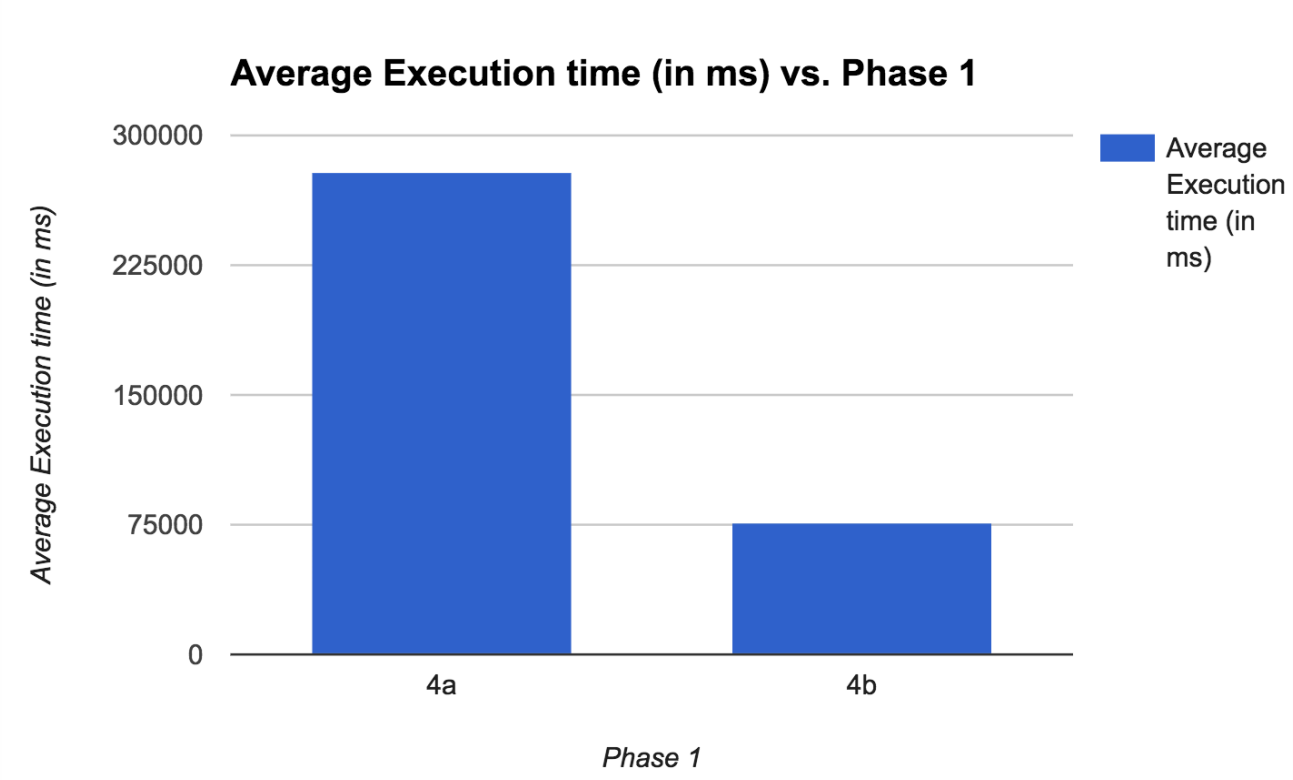


For the above queries, the query with index `3b` consumes more memory for constructing the index, but the computation is faster hence the execution time is smaller than the non-index query `3a`.

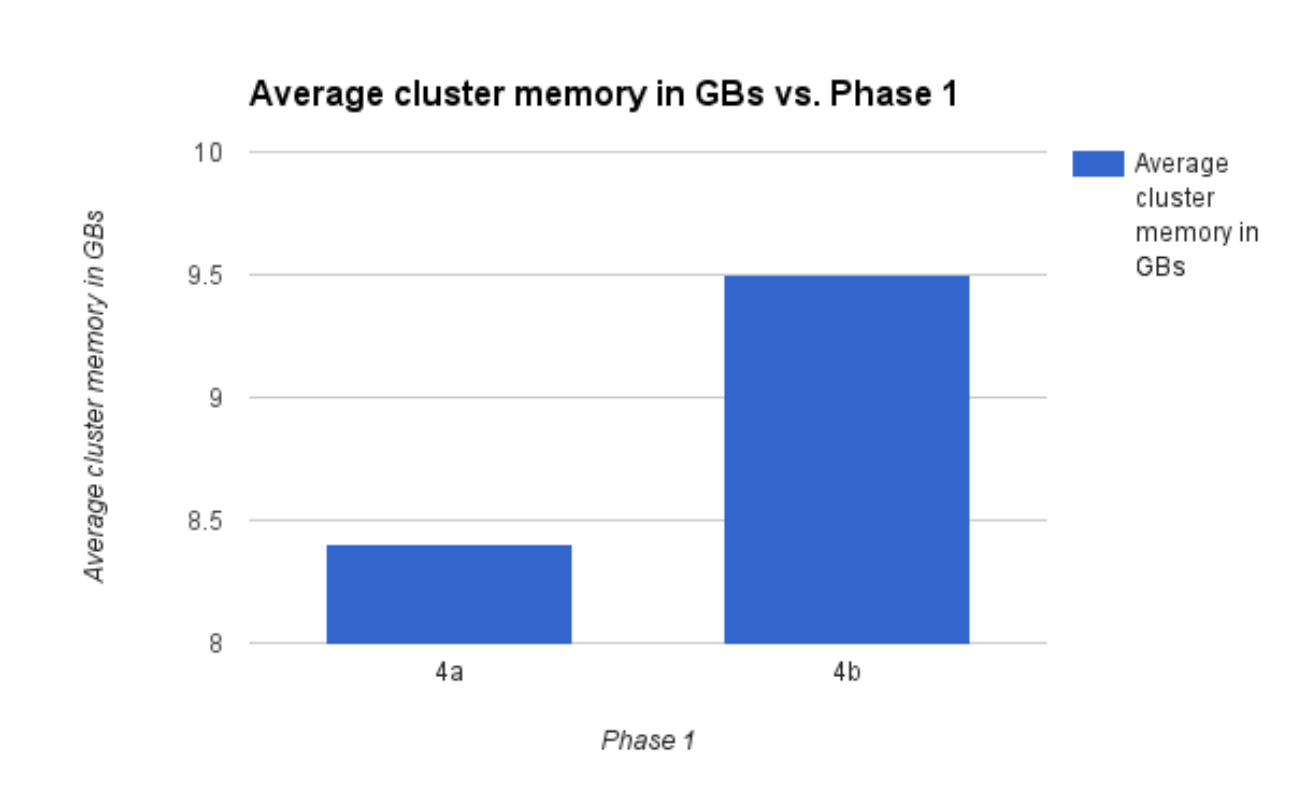


Comparison of 4a and 4b:

Phase 1	Average Execution time (in ms)	Average cluster memory in GBs	Average CPU usage %
4a	279419.3333	8.4	20.93333333
4b	76769.33333	9.5	18.93333333



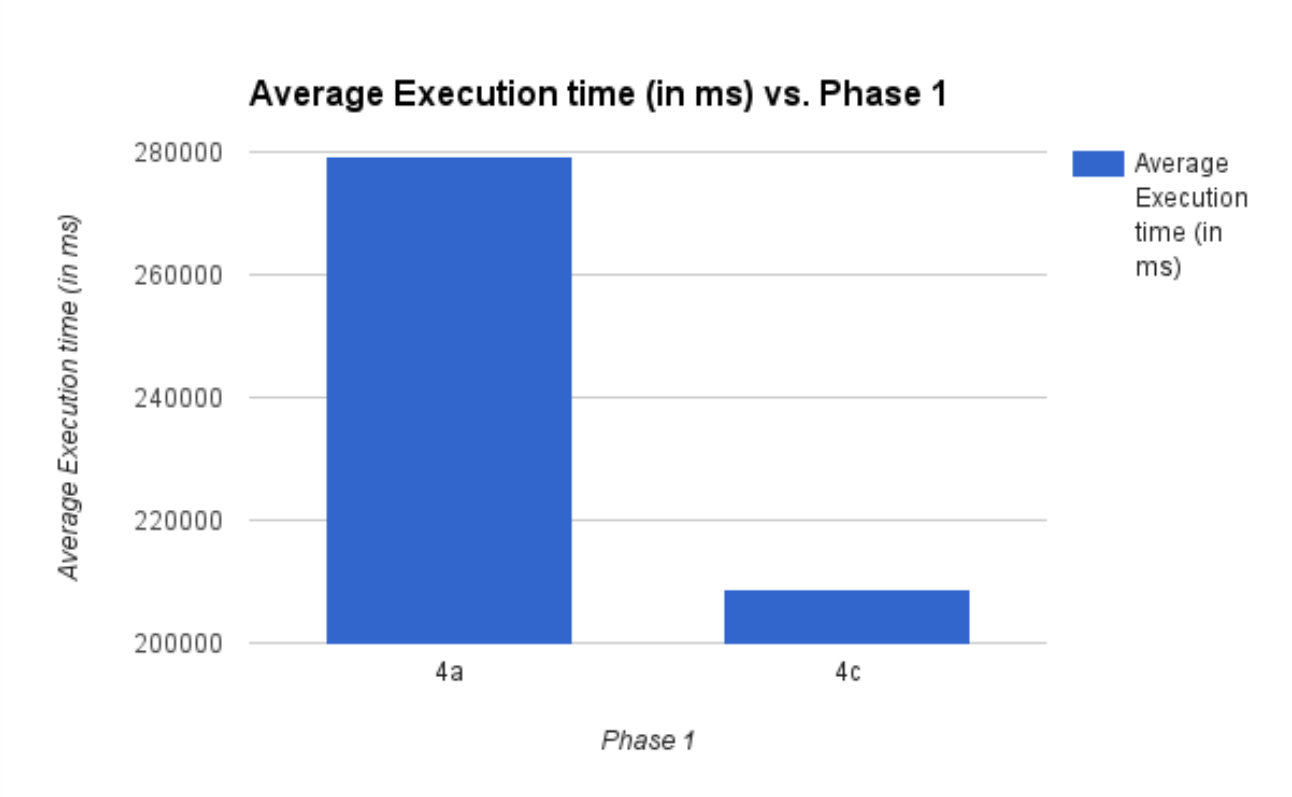
Query 4a is a spatial join query with equal grid partitioning and Query 4b is the same query with an RTree index on the pointRDD. The index speeds up the Spatial Join query as it would make 4b an index nested loop join instead of a simple nested loop which is what 4a would do.



On the memory front, building an index will consume more memory than the non-indexed join query. Hence we see an increase in memory usage on the cluster for 4b over 4a.

Comparison of 4a and 4c:

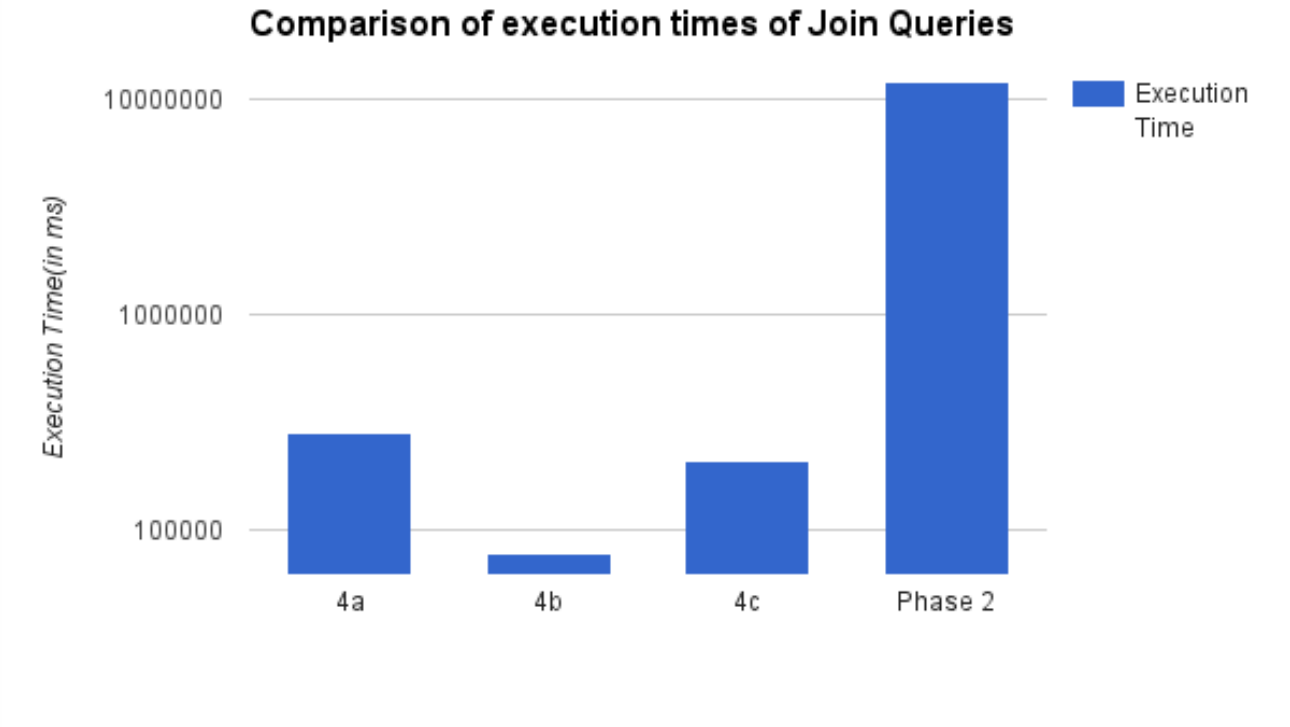
Phase 1	**Average Execution time (in ms)**	**Average cluster memory in GBs**	**Average CPU usage** *%*
4a	279419.3333	8.4	20.93333333
4c	208806	9.6	24.13333333



Query 4c builds an RTree partitioning over the pointRDD. While both queries are slower than query 4b which has an RTree index, query 4c is better in terms of execution time to query 4a. This is probably because the RTree partitioning has better access time to points during the nested loop join query than the equal grid, hence speeding up the query comparatively.

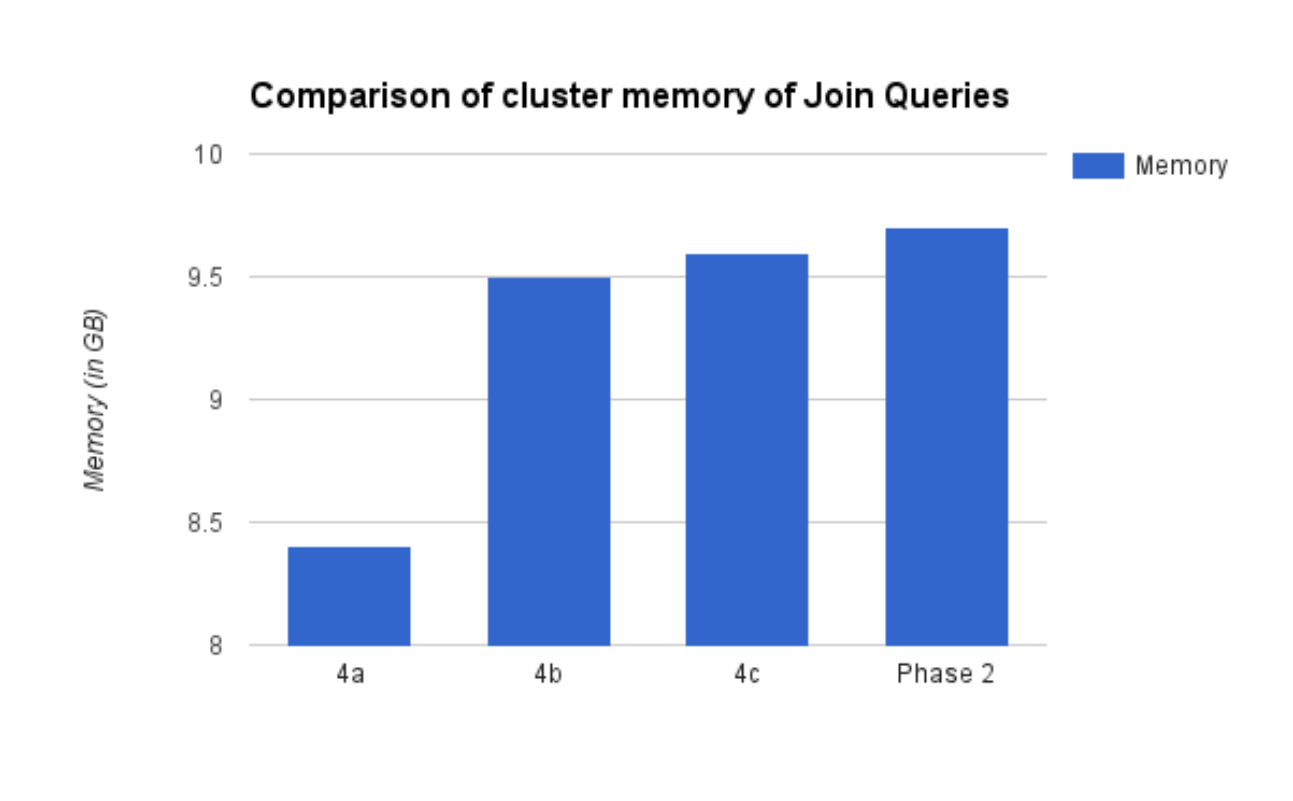
Comparison of Phase 2 cartesian join with Phase 1 4(a), (b), (c)

	4a	**4b**	**4c**	**Phase 2**
Execution Time(in ms)	279419.3333	76769.33333	208806	12021774.5



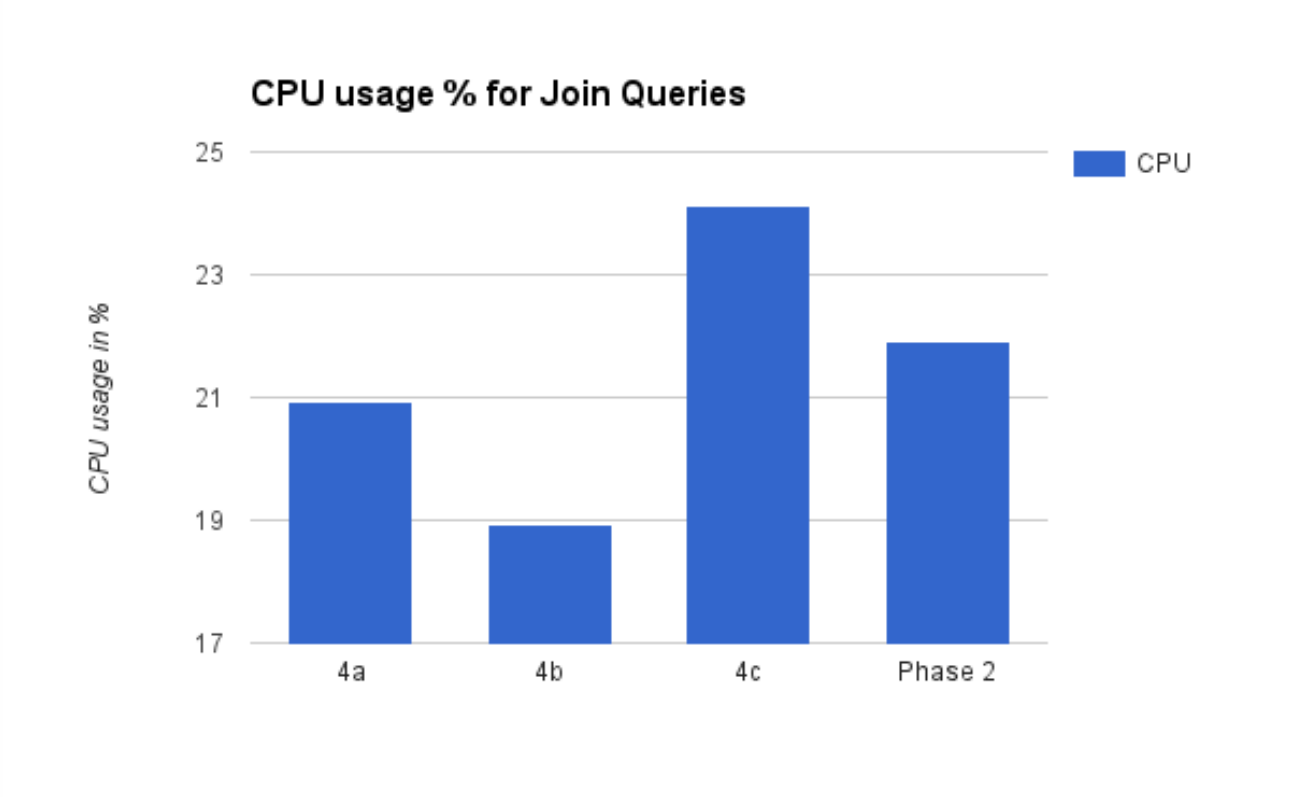
The execution time of the cartesian join query is almost 40 times slower than all the other join queries. This is due to the fact that the nested loop queries in 4a,4b,4c have grid partitioning which partitions the pointRDD and utilizes spark's parallelism, while the cartesian join query collects all rectangles and runs a spatial range query and further collects the pointRDD results. This slows down the spark processing since it doesn't utilize spark's parallelism.

	4a	**4b**	**4c**	**Phase 2**
Memory	8.4	9.5	9.6	9.7



The phase 2 query takes more memory on average than the phase 1 queries as it collects the results at the master. Also 4a takes less memory than 4b, 4c as it doesn't build an index like 4b and utilizes only the equal grid partitioning.

	4a	**4b**	**4c**	**Phase 2**
CPU	20.93333333	18.93333333	24.13333333	21.9



On the CPU front, the RTree indexed query 4b consumes the least CPU time among all queries.

IV. Team Members — Group # 25

1. Purushotham Kaushik Swaminathan [pswamin1@asu.edu] - 1208626480
2. ARAVIND RAJENDRAN [arajend6@asu.edu] - 1208684590
3. THAMIZH ARASAN RAJENDRAN [trajendr@asu.edu]- 1209319666
4. Balaji Chandrasekaran [bchandr6@asu.edu] - 1208948451
5. Suresh Gururajan Nolastname [sgurura3@asu.edu] - 1208567798

V. Summary

In this project, by analyzing the cluster CPU, Memory metrics and execution times of Spatial Range Query, Knn Query, Spatial Join query and Cartesian product based Join we find that — 1) queries that use indexes are faster albeit consuming more memory than regular queries that don't use indexes, 2) the cartesian join query is considerably slower than the nested loop queries that have grid partitioning.

VI. References

1. Jia Yu, Jinxuan Wu, Mohamed Sarwat. "GeoSpark: A Cluster Computing Framework forProcessing Large-Scale Spatial Data". (short paper) In Proceeding of the ACM InternationalConference on Advances in Geographic Information Systems ACM SIGSPATIAL GIS 2015,Seattle, WA, USA November 2015