Hot spot analysis on NYC Yellow Cab Taxi Dataset

Purushotham Kaushik Swaminathan, Suresh Gururajan Nolastname, Thamizh Arasan Rajendran,
Aravind Rajendran, Balaji Chandrasekaran

School of Computing, Informatics and Decision Systems Engineering, Arizona State University

Credits: ACM SIGSPATIAL Cup 2016, Problem Definition (Online) [3]

INTRODUCTION

We began this project by familiarizing ourselves with the GeoSpark API, obtaining run-time statistics for distributed computation techniques like Spatial Range Query, Spatial KNN query, Spatial Join query and their variations obtained by with and without joining the PointRDD using Equal grid or building R-tree indices. We successfully implemented the tasks on a Hadoop cluster running spark shell with geo-spark jar included. The second phase involved comparing the performance of R-tree indexing on PointRDD vs non-indexed PointRDD data. The third phase involved spatio-temporal analysis of a subset of the NYC yellow-cab taxi dataset (Jan 2015 data), to compute the Getis-Ord statistic[2] as a z-score to identify 50 hot spots within the mentioned grid.

Phase 1

Goal:

Set up a Hadoop cluster, running Apache Spark and performing the following spatial operations using GeoSpark API:

1.Spatial range query

2.Spatial KNN query

3.Spatial join query

Experimental Setup:

Three (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

Outcome:

We learnt how to spin a Hadoop cluster, run Apache Spark on top of it and perform spatial operations using GeoSpark. A demo is provided in [1].

Phase 2

Goal:

Perform spatial join query using simple Cartesian product algorithm. Compare the performance of various spatial join, range and nearest neighbor operations from Phase 1 on GeoSpark, running on a Hadoop cluster, using evaluation metrics such as memory usage, CPU usage and execution time using the Ganglia monitoring tool.

Experimental Setup:

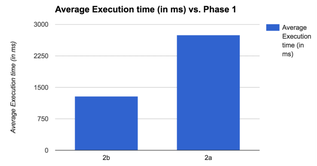
Run Phase 1's (1), (2) and (3) spatial operations on the same cluster as before, but this time, we additionally use Ganglia to monitor the cluster resource utilization for performance evaluation.

Results and Analysis:

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

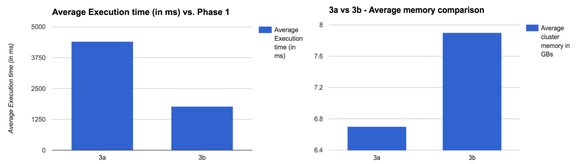
Figure 1. Comparison of performance of querying the RDD with and without using R-tree index on PointRDD

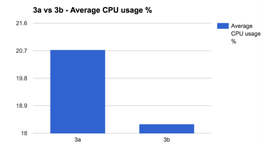
Here, we observe that querying using an R-tree index on PointRDD leads to faster execution, which is as expected.



In the second experiment, we compare the execution time of spatial KNN query without an R-tree index and the execution time using an R-tree index on PointRDD.

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

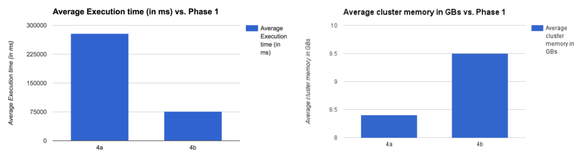




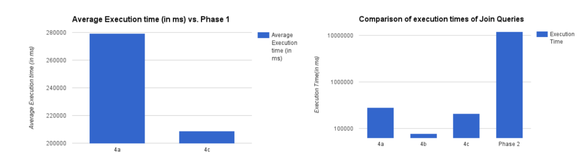
In the third experiment, we compare the execution time of spatial join query:

1. Using Equal grid without R-Tree index – (4a)
2. Using Equal grid with R-Tree index. – (4b)
3. Using R-Tree grid without R-Tree index. – (4c)

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.



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.

Phase 3

Goal:

Spatio-temporal analysis of a subset of the NYC yellow-cab taxi dataset (Jan 2015 data), to compute the Getis-Ord statistic[2] as a z-score and identify 50 hot spots within the mentioned grid.

Credits: ACM SIGSPATIAL Cup 2016, Problem Definition (Online) [3]

Figure [3] shows the distribution of NYC Yellow Cab Taxi Trip Records from January 2009 to June 2015. On the right, the figure shows source data clipped to an envelope encompassing (latitude 40.5N – 40.9N, longitude 73.7W – 74.25W)

Getis-Ord Statistic:

The Getis-Ord statistic [2] for a cell i is defined as:

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

where,

i = cell under consideration

j = neighboring cell

$w_{i,j}$ = spatial weight between cell i and j (1 if neighbor, 0 otherwise)

x_j = attribute value for cell j (in our case, the number of trips from a pickup location)

n = total number of cells

Approach using Spark:

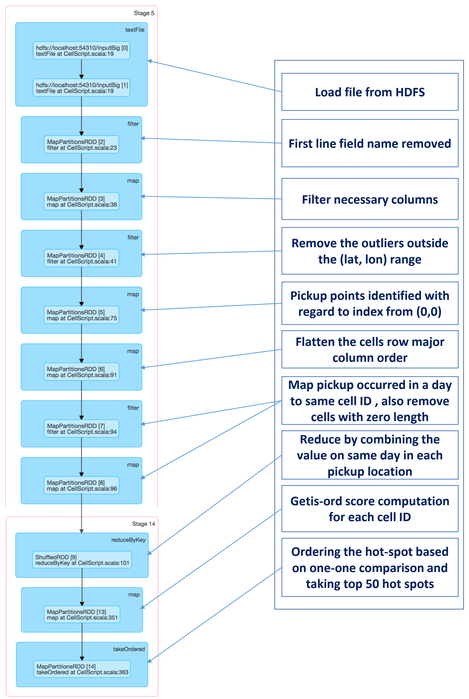


Figure: Flowchart showing our approach towards obtaining the Getis-ord statistics

Evaluation:

The dataset we used for the project was the Yellow taxi data from January 2015 the size of which was roughly 1.8 GB. The calculation of G* was performed on the same cluster as the one used in Phase 1 and 2 of the project.

We evaluated our code based on memory utilization, CPU and network cost.

Results and Analysis:

Our code was written in Scala [7], utilizing Spark RDDs to perform our computations faster. Five of the top hotspots and the corresponding locations in Google Maps are shown below:

((Cell X, Cell Y, Day ID), Z-score)

((4075,-7399,13),31.72195043617353)

((4075,-7399,14),31.528184290546584)

((4075,-7399,21),31.21857761734719)

((4075,-7399,12),30.8474941729361)

((4075,-7399,8),30.604127450536083)

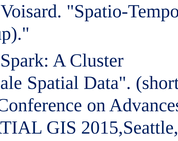


Figure. Top 50 hotspots on a heat map

We evaluated the total run time of the program calculated from the time of reading data from HDFS to the time of writing the results (i.e., the top 50 hotspots) back to HDFS and this whole process took approximately 3 minutes.

We attribute this performance primarily to code optimizations and Apache Spark's lazy transformations including the following:

1. All the tasks were distributed. For each task (refer flowchart), we move computation to nodes instead of materializing data for the next stages.
2. We perform the "collect" operation only once, to store a hash map of <Cell ID, Values> to be used in Getis-ord computations.
3. Since Spark doesn't write to disk between stages, we observed a significant speedup compared to Map-Reduce which writes to disk after any transformation/action.
4. Mehta et. al [4] discuss a distributed algorithm that optimizes the computation of G* by grouping cells and their neighbors in a data structure and in the same node in order to reduce the number of calls to the network required to bring the neighboring points' attributes for computing a cell's G* statistic.

   In our algorithm, we pre-collect the attributes for all the nodes and store it in a hash map which is then distributed to all nodes. Since the data set is 1.8 GB in size and our cluster's memory vastly exceeds this size, spillover of data into disk is avoided and all of the data can be kept in memory by Spark. This allows us to take advantage of the in-memory operation of Spark which greatly speeds up our program compared to Hadoop's Map Reduce.

REFERENCES

[1] Group 25 - DDS Project Phase 1, submission link: https://www.youtube.com/watch?v=ShfpMlbMFYo

[2] Ord, J.K. and A. Getis. 1995. "Local Spatial Autocorrelation Statistics: Distributional Issues and an Application" in Geographical Analysis 27(4).

[3] ACM SIGSPATIAL Cup 2016, Problem Definition (Online): http://sigspatial2016.sigspatial.org/giscup2016/problem Accessed: 12/04/2016

[4] Mehta, Paras, Christian Windolf, and Agnès Voisard. "Spatio-Temporal Hotspot Computation on Apache Spark (GIS Cup)."

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

[6] PowerPoint poster templates for research poster presentations, (Online), accessed: 12/5/2016 at 00:14 AM http://www.posterpresentations.com/html/free_poster_templates.html

[7] The Scala Programming Language, https://www.scala-lang.org/