Range Query and Skyline point in Spark and HDFS

Umesh Maroti Gaikwad, Nirmalya Gayen

1. Problem Description:

1.1 Range Query:

Given a set of point P with their dimension d, our goal is to find points in a given query range. $q \in P$ is in range if each respective dimension of q is in range $[x_1, x_1'][x_2, x_2'] \dots .[x_d, x_d']$ respectively.

Range query of points on single node system takes $O(log^d n)$ time using standard algorithm. Time complexity in distributed approach is explained in 4.1

1.2 Closest Pair of Points:

Given a set of point with their dimension, distance between each pair P and Q is calculated by using Euclidean distance formula for points with d dimension

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_d - q_d)^2}$$

Where, $(p_1, p_2, \dots, p_d)$ are the co-ordinates of point P and $(q_1, q_2, \dots, q_d)$ are the co-ordinates of point Q in d-dimensional space. A pair with shortest distance between them is a closest pair. There are multiple applications of closest pair of points such as Genetics, Travel, Hierarchical clustering, Greedy Matching, etc.

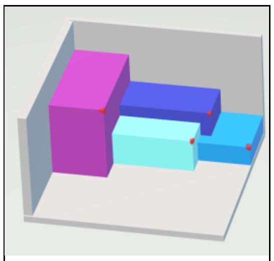
Finding closest pair of points on single node take $O(Nlog(N))$ time using standard algorithm. Time complexity in distributed approach is explained in 4.2

1.3 Skyline Points:

Skyline points are those points that will not get dominated by any other points in given data samples. Here domination means all the dimensions should be higher than respective dimensions of another point.

Given, a d-dimensional space, $D = (\delta_1, \delta_2, \dots, \delta_d)$, and a set of points $P = (p_1, p_2, \dots, p_n)$ is said to be a data set on D if every $p_i \in P$ is a d-dimensional data point on P. We use p_i^k to denote the k^{th} dimension value of point p_i

A point p_i is said to be dominate another point p_j if and only if $1 \le k \le d, p_i^k \ge p_j^k$. Set of points which are not being dominated by any other point from set P will be skyline set.



$Fig1$

$Fig1$ explains the skyline points in a $3D$ space.

Calculating skyline points in local machine takes $O(n.logk)$ time. But in a local machine with the increase in data points the n will increase. If we have p machines, we can divide give n/p points to each machine. Time to get skyline in each machine we reduced to $max\left(O(n/p \cdot logk_1), O(n/p \cdot logk_2), \dots, O\left(n/p \cdot logk_p\right)\right)$, Then we just need to merge the resulting skyline points. We will discuss this in Sec 4.3

2. Motivation:

2.1 Range Query:

Range Query has many applications in spatial database, GIS and also in any group of columns with orderable data type. For example:

- Query nearest restaurant in a map
- Query some specification range in e-commerce site.

2.2 Closest Pair of Points:

Find closest pair of points: Finding closest pair of point need to find shortest Euclidian distance between to points. Closest pair of points have different applications like Dynamic Minimum Spanning Tree, Collision Detection Applications, Travelling Salesman Heuristics, etc.

2.3 Skyline Points:

Find non-dominating points: Finding non-dominating points in the d-dimensional dataset has application in finding the overall best point in a dataset. It has application in GIS, Spatial Databases. For example:

- Large-scale spatial systems contain massive datasets with location information about the underlying objects. Range queries are often executed to extract information from specific parts of the address space.
- In an e-commerce site users might query for products in a range like a price in a range of [10,000, 30,000], specifications like etc, these queries need to be optimized +to work faster.

3. Related Work:

Big data computing tools such as Hadoop MapReduce, Apache Spark, etc. are widely adopted to build parallel data analysis applications to analyse big data. The target architecture for these tools is a distributed computing environment with multiple multi-core computing nodes. In [2], Eldawy et al. present a suite of scalable and efficient MapReduce algorithms for some of the key computational geometry problems such as convex hull, closest pair of points, etc., called CG Hadoop. Their experiments on a 25-machine cluster show that CG Hadoop achieves up to 14x and 115x better performance than traditional algorithms when using Apache Hadoop and SpatialHadoop (a Hadoop based system more suitable for spatial operations) systems respectively [2], [3].

Our work in this project lies in between above two categories. And we are focusing on some of the spatial algorithm's implementation mentioned above. It's not new approach but our goal is to implement these efficiently using MapReduce paradigm on indexed RDDs.

3. Problem Definition:

4.1 Range Query:

We will partition the points in p partitions (we don't need to group by any dimension value, it can be random).

Then pre-processing takes $O\left(nlog^{(d-1)}n\right)$ space and $O\left(\frac{n}{p}log^{d}\frac{n}{p}\right)$ time.

Then we can find all the points in a range in time $max\left(O\left(log^{d}\frac{n}{p}+k_1\right),\dots,O\left(log^{d}\frac{n}{p}+k_d\right)\right)$ by searching in each range.

Next, we need to merge the result.

4.2 Closest Pair of Points:

We will divide the point set in grid of $\mathbb{R}\times\mathbb{R}$. To partitions into grid, we will consider R equally spaced intervals along x-axis and R equally space intervals along y-axis. On every worker node it will calculate closest pair in that grid. It will be report closest pair for that grid and points near boundary. We will apply same algorithms on these reported points from each partition. Then the final result will be the closest pair of given points. Finding closest pair from N points take $O\left(Nlog(N)\right)$ time. Here are around N/R^2 points in each partition, so for each partition it will take $O\left(N/R^2log(N/R^2)\right)$ time.

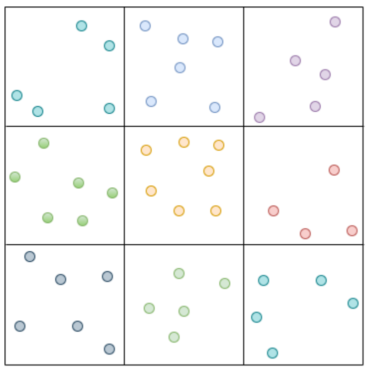
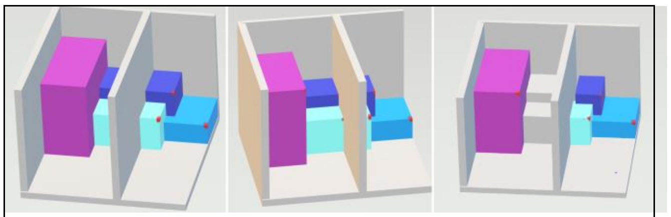


Fig: 3×3 Grid on given special data

4.3 Skyline Points:

In a local machine when we have all data points are accessible equally to all processing nodes. We can simply use skyline algorithm in $O(nlogk)$time.



$Fig3$

In $Fig3$ we have described the process of finding skyline points in a distributed environment.

(a) We will partition the points (we don't need to group by any dimension value, it can be random)
(b) We calculate sky-lines points in all the partitions.
(c) Next, we will merge the partitions. We can get some skyline points in some partitions, which would not be a skyline point after the merge.

5. Solution Approach:

5.1 Skyline Points:

For Skyline points our approach was to read all the points, from HDFS, next map the string data to points, Point is a POJO class, which represents single record of point. Next we are mapping all points in an LinkedList, here individual point represents a Skyline point. Now we have merged to skyline points using reduce(). For any two set of skyline points we peek two points from each of the sets and select the right most point. Remove all points bellow the selected point. Like this we get the merged skyline points which is sorted from right, in $O(n)$ time.

Expected number of skyline points in a set is O(k) the number of skyline point in the dataset.

Next we collect() the points from RDD.

5.2 Range Query:

For Range Query we read the points from HDFS, Then we partition the points in user-defined number of segments, using kay-value pair. Where the key is the boundary of the segment where the point lies in. Now we use mapByValue() to map the Iterable<Point> set to a QuadTree to query any segment on $O(\sqrt{n})$. As the number of segment is constant for a large data set. We can query the dataset in $O(\sqrt{n})$ time.

We have compared it with filter() method available in Spark.

5.3 Closest Pair of Points:

Similar to the previous, We read the points from HDFS, Then we partition the points in user-defined number of segments, using kay-value pair. Where the key is the boundary of the segment where the point lies in. Now we use mapByValue() to find the candidate points from Iterable<Point> set.

Next we merge the candidate points to find the minimum distance point.

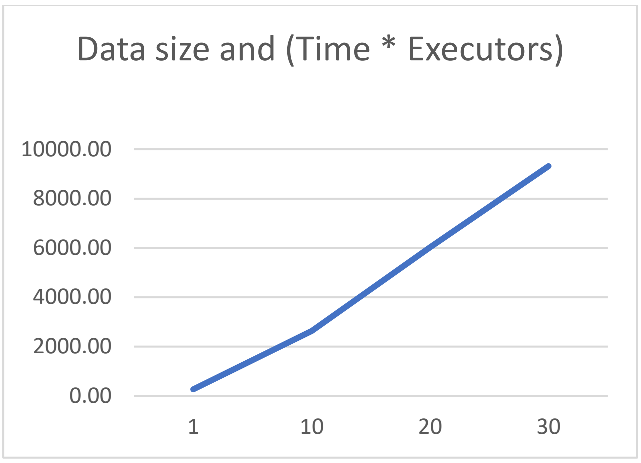
6. Experiments:

We did our experiments in Spark cluster with configuration: number of executors 10(for skyline) and 5(for all other algorithms), driver-memory 512m, executor-memory 1G, executor-cores 1.

6.1 Skyline Points:

We did our experiments with data set of 1M, 10M, 20M and 30M.

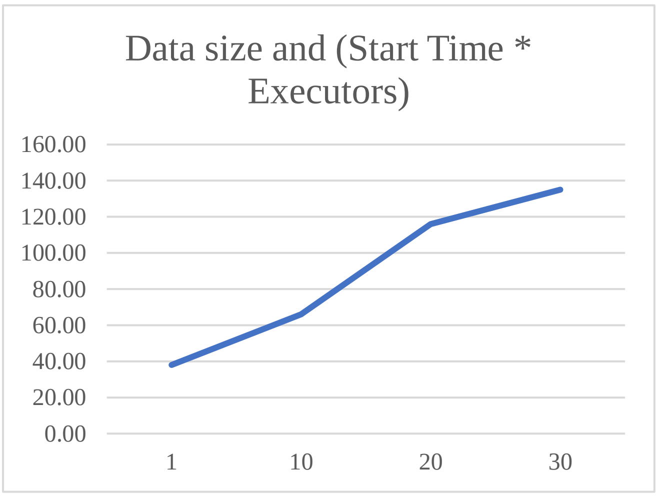
Data size (Million)	Time (Sec)	File Size (MB)	No. of Executors	No. of Exec. Provided
1	131	21	2	10
10	1318	206	2	10
20	1505	412	4	10
30	1863	617	5	10



6.2.1 Range Query (using filter()):

We did our experiments with data set of 1M, 10M, 20M and 30M.

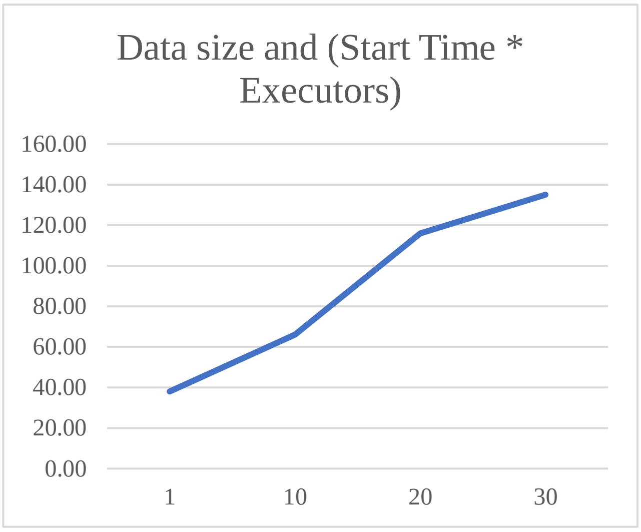
Data size (Million)	Start Time (Sec)	Query1 Time (Sec)	Query2 Time (Sec)	File Size (MB)	No. of Executors	No. of Exec. Provided
1	19	42392	13965	21	2	5
10	22	52382921	440681	206	3	5
20	29	84766851	1081383	412	4	5
30	27	77152154	9121300	617	5	5



6.2.2 Range Query (QuadTree):

We did our experiments with data set of 1M, 10M, 20M and 30M.

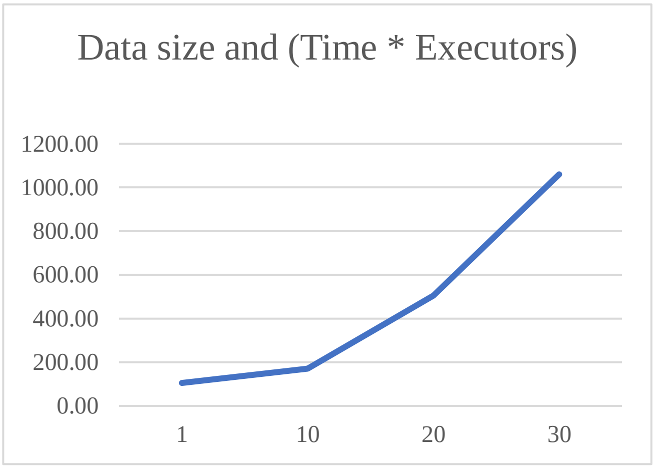
Data size (Million)	Start Time (Sec)	Query1 Time (Sec)	Query2 Time (Sec)	File Size (MB)	No. of Executors	No. of Exec. Provided
1	27	22392	23965	21	5	5
10	67	92382921	840681	206	5	5
20	80	184766851	1481383	412	5	5
30	73	227152154	22121300	617	5	5



6.3 Closest Pair of Points:

We did our experiments with data set of 1K, 1M, 5M and 10M.

Data size	Time (Sec)	File Size (MB)	No. of Executors	No. of Exec. Provided
1K	21	24KB	5	5
1M	34	21MB	5	5
5M	101	103MB	5	5
10M	212	206MB	5	5



7. Conclusion:

Skyline algorithm can be implemented, and it can be scale to any size as it required only reduce function which will compute Skyline point

Closest pair algorithm can give very large set of candidate points, which will cause overhead to master

Range Search algorithm in our approach time increase very less with increase in number of data point.

7. Future Work:

Many geospatial algorithms can be implemented in scalable fashion

Many Approximation algorithms are still need to be evolve to implement in scalable fashion

6. Reference:

[1] https://depts.washington.edu/dslab/MASS/reports/SaranyaGokulramkumar_thesis.pdf

[2] Ahmed Eldawy. CG Hadoop : Computational Geometry in MapReduce. Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 294–303, 2013.

[3] Ahmed Eldawy. SpatialHadoop : A MapReduce Framework for Spatial Data . 2015 IEEE 31st international conference on Data Engineering, 1:1352–1363.