Scalable Distance Joins for Point Data

Georgios Panagiotakopoulos
Department of Digital Systems
Big Data and Analytics
University of Piraeus
Piraeus
me2030@unipi.gr

ABSTRACT

Efficient distance Join query processing becomes critical as spatial data increases and the need to find spatial information multiplies. Questions of this type are computationally "accurate" because the criterion of matching two records is variable and is based on the calculation of the spatial distance of the points between them. This means that the join is done on a calculated field (spatial distance).

In order for spatial queries to be served in a short time, many technological approaches have been developed to execute them. The technology used in this work was Apache Spark, which is based on the principle of performing spatial queries on computer clusters. This approach leads to the division of the big problem into many smaller ones, the solution of which is assigned to cluster computers in parallel. This is how the computing power of multiple machines is exploited, resulting in the processing of work in a fraction of time.

This work implements a mechanism for searching geographical points which are randomly divided into two sets of points. The aim is to find the points of the two sets which are less than or equal to a value θ. The value of the distance of the points is calculated according to the Euclidean formula of distance. The application was written in Python using the PySpark library to run in the Apache Spark environment.

The application was executed in a cluster environment on the OKEANOS platform of the University of Piraeus. In order to evaluate the results, a comparison of program execution on a single computer and in the cluster environment was compared.

CONCEPTS

• Spatial Data • Distance join queries • Clustering • Parallel Processing

KEYWORDS

Apache Spark, nearest points, clustering

1 Field Overview

Today, the use of devices that are directly related to geographical location (such as mobile phones, IoT devices) produce large volumes of spatial data, in the sense that coordinates are recorded that are related to other data (for example, coordinates where there is traffic congestion). Such data is generated by both private devices and transport systems (vehicles or boats). Characteristic of spatial data is the speed in creating new ones, since the location is recorded with a high frequency by the tracking devices. The creation of such data in databases, creates a new category of applications, which processes spatial data and answers questions related to the distance of points between them. Indicative examples are: a) locating points close to the user (e.g. restaurants located 100 meters from the user's hotel), b) with data of two sets of points, to locate those points between the two sets that are less than a limit away (eg find the schools of Attica, which are closer to sports venues in Attica).

Research in the field of spatial data management has been under way for decades. Sowell et al. have dealt intensively with spatial joins in memory [1]. Recently, research has supported the use of Hadoop MapReduce for spatial data management [2]. Afrati and Ulman [3] formulated a framework for calculating a multi-join query in a single iteration. Liu et al. [4] studied the application of Voronoi diagrams to speed up k-nearest-neighbor queries using the MapReduce technique. Hadoop-GIS is an extension of Hadoop [5], which supports the execution of spatial queries in Hadoop using uniform grid indexes. Respectively, SpatialHadoop [6] creates generalized and local pointers to spatial data. At the same time, it has made modifications to the method of reading data from the Hadoop File System (HDFS), so that reading data is more efficient and in less time. The MDHbase [7] system is an extension of HBase to enable spatial queries and spatial data management.

The MapReduce technique gives good results for processing large volumes of data, while at the same time it has features of protection against software and hardware errors (fault tolerance). One issue with Hadoop MapReduce is that intermediate data generated during query processing must be written to HDFS, which slows down data processing speed, especially when an application requires multiple MapReduce processes. Apache Spark [8] (and other related systems such as Graphx [8], Spark-SQL [9] and DStream [10]) As a result, using memory to handle their processes, solves speed problems with MapReduce systems. This feature of Apache Spark gives it significant advantages in managing large spatial data. Many prototype systems and applications support processes in spatial data using Spark (such as GeoSpark [11], SpatialSpark [12], Magellan [13] and GeoTrellis [14]). However, there are open issues, which

have to do with the excessive creation of additional load on the network and the input / output units of the computer.

2 Overview of Apache Spark

Apache Spark started as a research project at the University of Berkeley, in the laboratory of big data and analytics.

Its purpose was to create a programming model that supports a wider range of applications than MapReduce, while at the same time being resistant to system hardware failures. MapReduce is not very efficient on queries that have short access time requirements and are processed in multiple parallel processes. Such applications are used in data analytics and include:

1. Machine learning algorithms and graph algorithms, such as PageRank.
2. Interactive data mining, where the user loads data into the memory of a computer cluster and asks repetitive queries.
3. Streaming management applications that aggregate measurements over time.

MapReduce is not optimized for this type of application because it relies on acyclic data flow. This means that an application must perform a series of distinct processes, each of which reads data from a secondary storage (such as a distributed file system), and rewrites the result in that secondary storage. This mechanism takes time to process at each step, as the data must be read and written to permanent storage.

Spark offers an abstract structure, named the Resilient Distributed Datasets (RDD), designed to efficiently support such applications. The main feature of RDDs is that during queries they are stored in memory, without the need to write them to a permanent storage unit. RDDs incorporate a mechanism that restores data in the event of a loss (such as a computer failure). This mechanism memorizes the way it was used to create the RDD (indicatively via map, join or group by transformations) and executes this method to recreate itself. RDD technology allows Spark to offer speeds up to 100 times higher than similar applications running MapReduce. Spark also integrates the SparkSQL language, which is an implementation of the SQL language of relational databases, so that queries to the system data can be easily compiled in the form of tables.

2 Problem Formulation

The problem to be solved by the application that will be developed in the context of this work is to identify pairs of points that belong to two different sets, which are at a distance less than one value that we set, and is called θ. In a conventional embodiment, an application must calculate for each point of a set (let Total A) its distance from all points of the other set (let Total B) and the pairs of points for which:

$$d\left(A_{(x_i,y_i)}, B_{(x_j,y_j)}\right) \leq \theta$$

where d is the distance between the two points, to be inserted into a new set R, which will be the result of the application.

The distance of the points is given based on the type of Euclidean distance:

$$d(A,B) = \sqrt{(B_x - A_x)^2 + \left(B_y - A_y\right)^2}$$

CSV files will be given as input to the application, which will have any columns of data, provided that the first column will necessarily contain the unique id of the point, the second the horizontal distance of the point from the beginning of the axis (longitude) and the third is its vertical distance from the beginning of the axes (latitude).

2 Implementation of application

The application was developed with the Python language and the use of the PySpark library, which allows distributed execution of the program on a cluster of connected computers in a Spark cluster architecture.

2.1 Algorithmic design

The basic idea of the algorithm applied to the implementation is to create a virtual "grid", which consists of square adjacent frames. The purpose of the application is to distribute the points of the two sets in the appropriate frames, in order to create the corresponding partitions which will then be processed by the Spark cluster. The algorithm is developed as follows:

1. The dimensions of the grid are calculated, calculating the smallest and largest values for the coordinates of all the points of A and B.
2. The resulting grid will have extreme points:
   $$\min longitude = \min(\min LonA, minLonB)$$
   $$\min latitude = \min(\min LatA, minLatB)$$
   and
   $$\max longitude = \max(\max LonA, maxLonB)$$
   $$\max latitude = \max(\max LatA, maxLatB)$$
3. The points of set A are distributed in the grid, according to their coordinates.
4. For each point of set B, it is distributed in a frame of the grid.
5. For this point, its distance from the points of its adjacent frames is calculated. If the distance of the point from a point adjacent to the frame is less than θ, then the point is copied to the corresponding adjacent frame.
6. When this process is completed, the two grids are merged into one, so that their frames are distributed to the workers of the Spark cluster and thus the search is completed in a fraction of the time that would be required if the algorithm was executed on a single computer.
7. For each partition, the distances of all points between them are calculated.
8. Pairs for which the Euclidean distance is less than θ are entered in the RDD of the result.

2.2 Development with PySpark

The application accepts as input data two CSV format files, which include spatial point data. Each of the input files should have the following format, in its first three columns: a) 1st column: unique point ID, b) the position of the point on the horizontal axis (longitude) and c) the position of the point on the vertical axis (latitude). There is no restriction on the title of these columns, the application internally renames the column names to obtain the predefined names required in queries).

Also, as input parameters two integers are required, which express the dimensions of the grid, in which the elements will be distributed. The product of these numbers will be equal to the number of partitions that the program will create and which will be processed by the available workers.

The application reads and processes each of the two input data files as follows:

a. The first file is read, and a Spark RDD is created, which has four columns (id, longitude, latitude, frame). Using SparkSQL, the coordinates of the extreme points are calculated:

result=sqlContext.sql("SELECT max(x) AS maxAx FROM set_a")

b. For each line of RDD, the corresponding point is distributed in the appropriate frame, according to its coordinates.

Figure 1 shows the mechanism that distributes the elements of set A within the grid. The points belonging to each frame will be included in the corresponding partition, which will be sent to the Spark worker for processing.

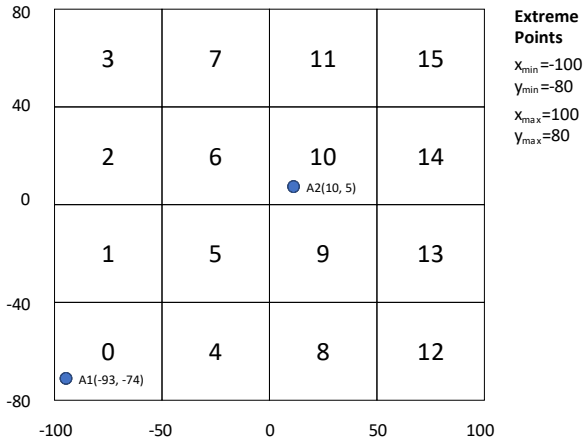

Figure 1. Mechanism of distribution of points within the total grid

After the distribution of the first dataset is completed, then the distribution of the points of the second dataset in the grid begins. For each point that is distributed in a frame of the grid, its distance is calculated with the points of its adjacent frames, in order to determine those points of the adjacent frames, which are less than or equal to θ. If there are such points, then a single copy of the point is made in the corresponding adjacent box (Figure 2).

At this point the processing of the points of the two sets has been completed and the system is ready to perform the query of calculating the distance of the points of each frame.

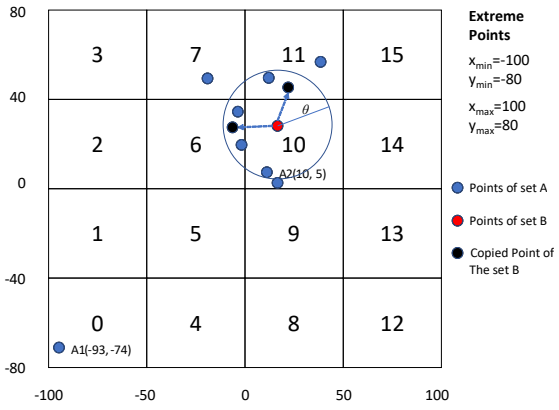

Figure 2. Mechanism for copying points of set B in adjacent frames, when there are points in them of set A at a distance ≤ *θ*

Each frame is assigned to a cluster worker for processing. SparkSQL executes a distance join query between points in two different sets that belong to the same context. The SparkSQL query running on each partition created is as follows:

```
SELECT A.id, B.id
        FROM
        (   SELECT  set_a.id AS id, set_a.x AS x, set_a.y as y
            FROM set_a
            WHERE dataset = 'A') AS A
            INNER JOIN
        (   SELECT set_a.id, set_a.x AS x, set_a.y as y
            FROM set_a
            WHERE dataset = 'B') AS B
            ON SQRT(POWER(B.y-A.y, 2) + POWER(B.x-A.x,
2)) <= θ
```

The result of the question is the pairs of points of set A, which are at a distance less than or equal to the value *θ*. Each worker runs the program independently on the data of the respective partition and calculates the results corresponding to this data.

The result of the processing of the different workers is returned to the Spark master, who combines the results.

Results

To measure the performance of the application using Apache Spark, Python and Spark SQL, one test was performed, with two data sets. Each set includes 500,000 points. In addition, in order to make a realistic comparison of Spark technology with conventional relational databases, an execution experiment was performed by

creating two relational tables and executing a SQL JOIN query on their coordinate columns.

TABLE 1. EXECUTION RESULTS

Execution platform	*Partition creation time*	*Neighborhood query execution time*
Microsoft SQL Server (locally)	-	12h:08min
Apache Spark (pySpark) application locally	1.637,909 sec	0,02293 sec
Apache Spark (pySpark) application (OKEANOS)	5.723,910 sec	0,18629 sec

The SQL query and the Spark application were executed locally on an Intel core i7 computer with 32GByte RAM and SSD drive.

REFERENCES

This work was done in the course "Big Data Processing - Techniques and Tools" of the MSc Information Systems and Services with specialization in Big Data and Analytics of the University of Piraeus and supervising Professor Mr. Doulkeridis.

BIBLIOGRAPHICAL REFERENCES

[1] V. Gaede and O. Gunther, "Multidimensional access methods," *ACM Comput.,* 1998.

[2] B. Sowell, M. Vaz Salles, T. Cao, A. Demers and J. Gehrke, "An experimental analysis of iterated spatial joins in main memory," *VLDB,* 2013.

[3] F. N. Afrati and J. D. Ullman, "Optimizing joins in a map-reduce environment," National Technical University of Athens, Stanford University, 2009.

[4] W. Lu, Y. Shen, S. Chen and B. Ooi, "Efficient processing of k nearest," *VLDB,* 2012.

[5] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang and S. Saltz, "Hadoop GIS: A high performance spatial data warehousing system," *VLDB,* 2013.

[6] A. Eldawy and F. B. Mokbel, "SpatialHadoop: A MapReduce framework for spatial," *ICDE,* 2015.

[7] S. Nishimura, S. Das, D. Agrawal and A. E. Abbadi, "MD-HBase: A scalable multi-dimensional data infrastructure for location aware services," *MDM,* 2011.

[8] M. Zaharia, "An Architecture for Fast and General Data Processing on Large Clusters," *Association for Computing Machinery and Morgan,* 2016.

[9] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," *OSDI,* 2014.

[10] DataFlair, "Apache Spark DStream (Discretized Streams)," 2021. [Online]. Available: https://data-flair.training/blogs/apache-spark-dstream-discretized-streams/.

[11] J. Yu, J. Wu and M. Sarwat, *ACM SIGSPATIAL,* 2016.

[12] S. Hagedorn, P. Götze and K. Sattler, "Big Spatial Data Processing Frameworks: Feature and Performance Evaluation," in *International Conference on Extending Database Technology (EDBT)*, Venice, Italy, 2017.

[13] N. Jordan and R. Sriharsha, "Magellan: Geospatial Analytics Using Spark," 2018. [Online]. Available: https://github.com/harsha2010/magellan.

[14] GeoTrellis, "GeoTrellis is a geographic data processing engine for high performance applications," GeoTrellis, 2020. [Online]. Available: https://geotrellis.io/.

Conference Name: HFBG'21
Conference Short Name: HFBG'21
Conference Location: Piraeus, Greece
ISBN:111-1-1111-0000-0/21/05
Year:2021
Date: May

DOI:10.1145/1234567890
RRH: