Scalable Query Processing on Big Spatial Data

Thanasis Georgiadis,
supervised by Nikos Mamoulis

University of Ioannina, Greece

Abstract

We live in the era of big data. Huge amounts of complex information, such as spatial data, are being generated daily by organizations, the industry and individuals at an increasing rate. At the same time, distributed and cloud computing systems are becoming more efficient, more available and with cheaper memory than ever. Thus, they are being established as the leading tool for big data management.

The goal of this doctoral work is to research and design novel scalable techniques that will support fast query processing on spatial data in a distributed environment. First, we aim to create new techniques that will simplify operations performed when processing complex spatial objects such as polygons, while simultaneously retaining result accuracy. Additionally, we study memory management mechanisms for individual machines, while optimizing size and performance trade-offs. Finally, we discuss the challenges of a distributed environment and provide ways to use optimized spatial data management techniques while ensuring scalability.

Keywords

Spatial Data, Distributed Systems, Query Processing, Scalable Techniques

1. Introduction

Spatial data management has been widely studied for the last forty years [1, 5], with an increased interest lately due to the large amounts of data that are generated daily. Spatial data refers to geometrical objects such as points, lines, polygons etc. that can be used to represent real world entities like buildings, lakes or points of interest (restaurants, banks etc.). They are usually generated and handled by Geographic Information Systems (GIS) that have to effectively organize and manage large quantities of such data, in order to use it commercially and scientifically.

Some of the most commonly used query types on spatial data include intersection joins (pairs of objects that intersect) [2], within joins (pairs of objects that the first is completely inside of the second), range queries (objects that intersect a given rectangular or polygonal area), distance joins (pairs of objects with less than a specified maximum distance between them) and k-nearest neighbors (the k geographically closest objects to the target object).

Resolving spatial queries is usually done in two steps: the *filter* and the *refinement* stages [1]. The first step filters a large portion of the data through simple operations, so that only a small amount of candidates will move on to the expensive refinement phase. In the second step, the objects are geometrically checked with one another to identify their relationship, which can be proved costly for complex geometries such as polygons.

When dealing with such geometries, the filter step usually processes some approximation of the objects (instead of their original geometries) and they are usually stored into hierarchical indices such as R-tree [3] or quadtree [4]. For example, polygons can be processed roughly using their Minimum Bounding Rectangles (MBRs), which are far more simpler to operate upon. There have been many approximations throughout the years (Convex Hull, 5C, Maximum Enclosed Rectangle and more) [5, 6], each one with its own attributes and various applicability on different query types.

Additionally, the concurrent progress and increased availability of distributed and cloud computing systems in the last couple of decades, has rendered them a powerful tool for big data management [7]. Techniques that partition the data to worker nodes which in turn can process it independently and simultaneously with one another, enable processing huge volumes of data in an environment with larger collective memory.

2. Motivation & Challenges

The major problem in spatial query evaluation, is that spatial relationships between complex geometries such as polygons, are expensive to identify, because they include edge intersection detection

Published in the Workshop Proceedings of the EDBT/ICDT 2023 Joint Conference (March 28-March 31, 2023, Ioannina, Greece)

EMAIL: ageorgiadis@cs.uoi.gr (T. Georgiadis)

and point-in-polygon tests (for intersection and containment, respectively). Thus, being able to reduce the total refinement workload is crucial when attempting to improve query evaluation time.

Additionally, tree-based spatial indices are not fit for highly dynamic data due to their large update cost. Approximations other than the MBR, are usually not stored in such indices. Instead, they are used in an *intermediate filter* right after the MBR-filtering stage, to further reduce the refinement candidates. However, not all approximation perform well and they are usually able to detect either true hits or true negatives, but not both.

Distributed commercial and scientific GIS (Apache Sedona, Spatial Hadoop, SIMBA, Magellan), cloud data warehousing services (Amazon Redshift) and companies that generate and use spatial data daily for their services (Twitter, Uber, Google), have an increasing demand for fast and scalable spatial data processing.

To increase parallelism, distributed systems must partition the data in a way that allows as much of the work as possible to be done independently by the worker nodes. For different data types however, this adds new challenges and is not always clear what partitioning method performs best for each query type. Geometrical approximations can be useful for data homogeneity and detecting true negatives faster, but they can not always identify true positives so they tend to lack in accuracy.

Additionally, most spatial data management systems focus on partitioning the data effectively, with little to no optimization done on a single machine. Our ultimate goal is to study scalable ways to process spatial data in a distributed environment and to face the challenges it establishes.

3. Scalable Query Processing

In this section we discuss the main goals of this doctoral research and how we deal with the previously mentioned challenges. It focuses on two individual aspects of distributed big data management: the techniques of processing queries on spatial data and the application of such methods in a distributed environment, while ensuring scalability.

The performance of single node query processing can benefit from novel indexing techniques and specialized data type-focused approaches [5, 8, 9], as well as certain polygon approximations that can speed up the query evaluation by reducing the amount of objects that need to be geometrically refined.

3.1. Raster Intervals Polygon Approximation

We have created a novel approximation technique for polygons called *Raster Intervals*(RI) [8], which has been proven to perform well in intersection and within joins. Additionally, it has the prospect to work well for range queries in which the window may not only be a rectangle, but also a complex polygon.

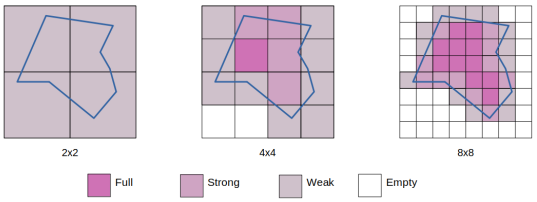

Figure 1: How grids of different granularity create variously detailed approximations.

RI are created through a pre-processing of the original data by rasterizing each object using a global grid and then ordering each cell using the Hilbert Curve [10]. Grid granularity not only affects the detail of each approximation but also the size in bytes it requires in memory (Figure 1). A granularity of $2^{16} \times 2^{16}$ is usually enough, however depending on data distribution and space, more detailed grids may be necessary. Then, consecutive cell identifiers are merged to create the intervals. Additionally, a cell may be only partially covered by the original geometry, so extra information about each cell has to be taken into account in order to use the approximation for accurate results.

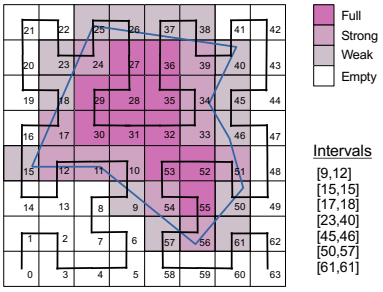

Figure 2: The Hilbert curve cell enumeration and interval generation for a polygon in a 8×8 space.

Inspired by the Raster Approximation [11], we use a modified classification for each cell, depending on the percentage of its area that overlaps with the original geometry: *Full* (100%), *Strong* ($> 50\%$),

Weak ($\leq 50\%$), as seen in Figure 2. This means that only *Strong-Weak, Weak-Strong and Weak-Weak* cases need to be further refined, while the rest of the cases can be safely either accepted or rejected.

In order to store this information in a compressed manner, a unique binary code is assigned to each case, depending on which one of the two data sets is currently being processed (Table 1). We then glue together the cell codes for each interval, creating a single *interval coding* for it. Therefore each object is now represented by a set of intervals and a set of their respective codings.

	input R	**input** S
full	011	101
strong	101	011
weak	100	010

Table 1
3-bit type codes for each input dataset

3.1.1. Compression

RI may generate many intervals for large polygons or high granularity, which results in too much memory usage. By compressing the interval array we can save space, with low decompression overhead and without affecting the evaluation time too much. Variable byte compression (Vbyte) is extremely popular and has been proven to achieve good ratio and decompression speed [12, 13]. It is a byte-aligned compression scheme for integers in which seven bits of a byte are used to represent a number and the eighth bit is used to indicate whether or not more bytes follow for that same number (1 if yes, else 0). The decompression is performed by removing the least (or in some versions the most) significant bit and concatenating the septets of bits together.

By employing SIMD instructions for faster binary operations, Vbyte can achieve even better performance for large arrays of integers [14]. However they are not particularly useful in our case, since an object will usually have a number of intervals ranging from dozens to a few hundreds, unless created using a very high granularity grid (larger than $2^{17} \times 2^{17}$).

3.1.2. Intersection/Within Joins

The RI approximation can be used in intersection/within joins for real world applications. Instead of comparing the original geometries of polygons, which as described previously is really expensive, possible interval overlaps are detected and their codes are compared to assess the situation between two objects. However, this method is used as an intermediate filter in the pipeline, rather than the first step of the whole process. It still depends on an MBR filter such as the Two-Layer Partitioning [9] to discard pairs of irrelevant objects.

To tackle the unavoidable added cost of the intermediate filter, we will also investigate a modified version, in which we separate and duplicate the main intervals (ALL-intervals), so that we have any Full-Strong-only (FS-intervals) and Full-only (F-intervals) cases as separate sets. Then, after detecting an overlap in a pair's ALL-intervals, the filter will check for overlaps in the SF-intervals, followed by the ALL-F and F-ALL types of intervals respectively for each set.

3.1.3. Range Queries

RI can also be used in range queries on polygonal data, by creating the RI approximation of the selection window and effectively joining it with a collection of objects. Instead of comparing the window's shape with the actual polygonal geometry, we detect interval overlaps between the window's and the polygons' approximations. Since the performance of a range query increases in relation to the window's shape and complexity, the RI approximation works pretty well for complicated, polygonal windows by reducing it into a set of integer tuples.

3.2. Distributed Query Processing

One of the major challenges found in distributed spatial data management, is data partitioning. The worker nodes must be able to process queries on their assigned data with minimal dependence to data partitioned across the other nodes.

3.2.1. Partitioning

Grid. Grid partitioning works quite well, since it assigns different sections (cells) of the total geographic area to nodes and they only process objects that lie on their designated areas. Nearby cells are usually assigned to different worker nodes, so that objects concentrated in that area (and are thus likely to participate as pairs in a query) are distributed to more than one worker nodes, enabling the system to utilize more of its resources in each case.

Trees. Similarly to a grid, trees can also be used to partition data. KD-trees [15] for points or R-trees [3] for polygons successfully divide space and same-level areas can be partitioned across a collection of nodes, so that each worker works on a specified data region. However, load balancing on tree-based

partitions is greatly depended on query type. For example, in intersection joins it might work well since the participating data sets are joined in their entirety. But in range queries, where a certain area is specified, all the workload might be put on a single machine, depending on data distribution.

Duplicate Detection. In both partitioning methods discussed, geometries usually intersect more than one of the designated sections, so duplicate entries are distributed across the system and may affect the final result and the total processing time. A duplicate detection mechanism must exist and it is usually done by either assigning each object to exclusively one section [16] through a pivot point or by minimizing the amount of checks for each object during the query processing[9].

Approximation Partitioning Considering that the de-duplication mechanism usually takes place in the MBR filter, we will focus on scalable intermediate filters for different types of queries and spatial data, since the refinement phase is what takes up 99% of the total evaluation time has the greatest room for improvement. For example, each node can use the Raster Intervals approximation in an intermediate filter, independently of how the data has been partitioned (i.e. the intermediate filter does not include the de-duplication mechanism). Additionally, we will investigate how well the grid partitioning performs for different approximations and queries, since it achieves considerably better workload balancing than trees.

4. Conclusion

To sum up, in this work we focus on the main issues that big spatial data management has to battle in order to be scalable. We discussed techniques that help preserve individuality and independence whilst preserving good throughput at query processing. In the future, more data types such as points and lines (which are typically not approximated) will be studied in order to be supported by these partitioning techniques and query processing methods, to improve system applicability.

5. Acknowledgments

T. Georgiadis was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the "2nd Call for HFRI Research Projects to support Faculty Members & Researchers" (Project No. 2757).

References

[1] N. Mamoulis, Spatial Data Management, Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2011.

[2] E. H. Jacox, H. Samet, Spatial join techniques, ACM Trans. Database Syst. 32 (1) (2007) 7.

[3] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: B. Yormark (Ed.), SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984, ACM Press, 1984, pp. 47–57.

[4] R. A. Finkel, J. L. Bentley, Quad trees: A data structure for retrieval on composite keys, Acta Informatica 4 (1974) 1–9.

[5] T. Brinkhoff, H. Kriegel, R. Schneider, B. Seeger, Multi-step processing of spatial joins, in: Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, USA, May 24-27, 1994, ACM Press, 1994, pp. 197–208.

[6] T. Brinkhoff, H. Kriegel, R. Schneider, Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems, in: Proceedings of the Ninth International Conference on Data Engineering, April 19-23, 1993, Vienna, Austria, IEEE Computer Society, 1993, pp. 40–49.

[7] V. Pandey, A. Kipf, T. Neumann, A. Kemper, How good are modern spatial analytics systems?, Proceedings of the VLDB Endowment 11 (2018) 1661–1673.

[8] T. Georgiadis, N. Mamoulis, Raster intervals: An approximation technique for polygon intersection joins, in: Proceedings of the 2023 ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, June, 2023.

[9] D. Tsitsigkos, K. Lampropoulos, P. Bouros, N. Mamoulis, M. Terrovitis, A two-layer partitioning for non-point spatial data, in: 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021, IEEE, 2021, pp. 1787–1798.

[10] D. Hilbert, Über die stetige Abbildung einer Linie auf ein Flächenstück, 1970, pp. 1–2.

[11] G. Zimbrao, J. M. de Souza, A raster approximation for processing of spatial joins, in: VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA, 1998, pp. 558–569.

[12] D. Lemire, L. Boytsov, Decoding billions of integers per second through vectorization, Softw. Pract. Exp. 45 (1) (2015) 1–29.

[13] H. E. Williams, J. Zobel, Compressing integers for fast file access, Comput. J. 42 (3) (1999) 193–201.

[14] D. Lemire, C. Rupp, Upscaledb: Efficient integer-key compression in a key-value store using SIMD instructions, Inf. Syst. 66 (2017) 13–23.

[15] J. Bentley, Multidimensional binary search trees used for associative searching, communications of the ACM September, 1975. vol. 18: pp. 509-517 : ill. includes bibliography. 18.

[16] J.-P. Dittrich, B. Seeger, Data redundancy and duplicate detection in spatial join processing, 2000, pp. 535 – 546.