Raptor: Large Scale Analysis of Big Raster and Vector Data

Samriddhi Singla
Ahmed Eldawy
UC Riverside
Riverside, CA, USA
{ssing068,eldawy}@ucr.edu

Rami Alghamdi
University of Minnesota, Twin Cities
Minneapolis, MN, USA
algha017@umn.edu

Mohamed F. Mokbel*
Qatar Computing Research Institute
Doha, Qatar
mmokbel@hbku.edu.qa

ABSTRACT

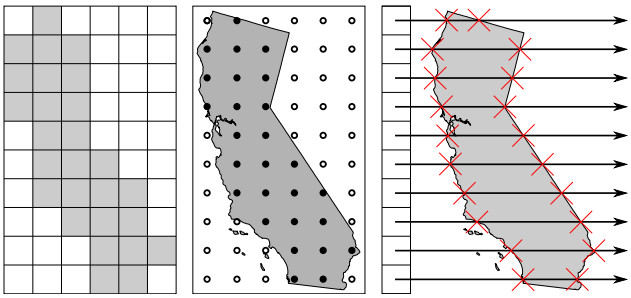
With the increase in amount of remote sensing data, there have been efforts to efficiently process it to help ecologists and geographers answer queries. However, they often need to process this data in combination with vector data, for example, city boundaries. Existing efforts require one dataset to be converted to the other representation, which is extremely inefficient for large datasets. In this demonstration, we focus on the zonal statistics problem, which computes the statistics over a raster layer for each polygon in a vector layer. We demonstrate three approaches, vector-based, raster-based, and *raptor-based* approaches. The latter is a recent effort of combining raster and vector data without a need of any conversion. This demo will allow users to run their own queries in any of the three methods and observe the differences in their performance depending on different raster and vector dataset sizes.

PVLDB Reference Format:
Samriddhi Singla, Ahmed Eldawy, Rami Alghamdi and Mohamed F. Mokbel. Raptor: Large Scale Analysis of Big Raster and Vector Data. *PVLDB*, 12(12): 1950-1953, 2019.
DOI: https://doi.org/10.14778/3352063.3352107

1. INTRODUCTION

The rapid advancement in remote sensing technology has led to a tremendous increase in the amount of spatial data collected in various domains. For example, the NASA EOSDIS archive contains more than 17 petabytes of data with an expectation to grow to 330 petabytes by 2025. Similarly, the Sentinel-1A satellite collected five petabytes in two years and is expected to continuously work until 2030.

This growth urged many researchers to build new systems for big spatial data including SpatialHadoop [5], GeoSpark [12], Simba [11], SciDB [10], RasDaMan [2], and GeoTrellis [8]. However, all these systems focus on processing either big raster data [2, 8, 10], such as satellite images, or big vector data [5, 11, 12], such as map data or geo-tagged objects. While these systems are very efficient in their core function, they provide a poor performance for queries that combine both vector and raster data.



(a) Raster (b) Vector (c) Raptor

Figure 1: Processing models

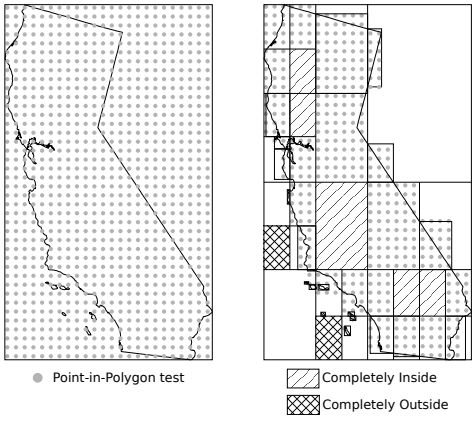
This demonstration focuses on the *zonal statistics* problem, which aggregates all the values from the raster layer that overlap with a set of polygons, e.g., computes the average temperature for each state in the US. This query has many applications including the study by ecologists of the effect of vegetation and temperature on human settlement [7] and by geographers for analyzing terabytes of socio-economic and environmental data [6]. Traditional systems for big spatial data follow one of two approaches, *rasterize* and *vectorize*, as illustrated in Figures 1(a) and 1(b). The *rasterize* method converts each polygon to a set of pixels in a raster layer and runs an overlay method to combine it with the raster layer. On the other hand, the vectorize method converts each pixel in the raster layer to a point and runs a point-in-polygon query. As the sizes of the raster and vector layers increase (i.e., up to trillions of pixels and hundreds of millions of line segments), the conversion step becomes very expensive and throttles the performance of the query.

Recently, a new processing model, termed Raptor[1], emerged as a more efficient method for queries that require the concurrent processing of raster and vector data. For example, the *scanline* method [4], illustrated in figure 4, evaluates the zonal statistics problem by first finding the intersections between the polygons and the raster data and then using these intersections to compute the desired aggregate functions, e.g., average. The key idea in this approach is that it avoids the conversion step by making the core query processor aware of the characteristics of the raster and vector

* Also affiliated with University of Minnesota, MN, USA.

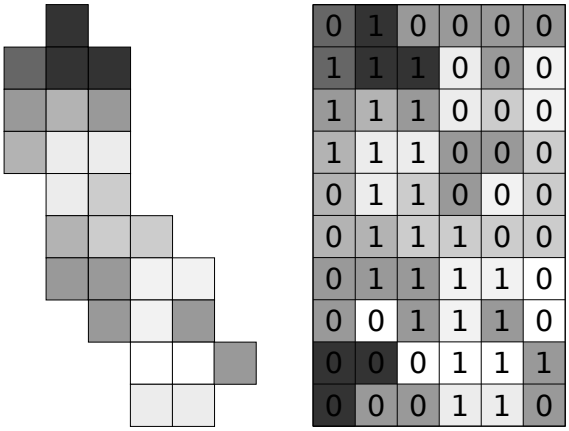
Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352107

[1]Raptor stands for RAster Plus vecTOR



(a) Naïve (b) QSplit

Figure 2: Vector-based Methods



(a) Clipping (b) Masking

Figure 3: Raster-based methods

representations. For example, it minimizes the disk access by following the representation format of the raster layer, i.e., row-major, column-major, or tiled representation.

This demonstration provides a comparison test bed for the three processing methods, namely, *raster*, *vector*, and *raptor*. The demonstration will use world-wide satellite data for temperature and vegetation over a course of five years (2014-2018). This big raster data will be combined with a vector data that represents all countries, states, counties, and cities (or their counterparts) in the entire world. The attendees will be able to get answers for questions like: "*What was the average temperature in my home city over the last month?*", or "*How did the average vegetation change in my country over the last year?*" In addition to the query answer, the demonstration will help database researchers get deeper insight of the results by providing several performance metrics, e.g., computation time, memory, and IO, for the demonstrated methods to express the power of the three approaches.

2. TECHNIQUES

This section provides a quick overview of the processing approaches for the zonal statistics problem. This work is categorized in three categories, vector-based approaches, raster-based approaches, and raptor-based approaches.

2.1 Vector-based Approaches

In this section, we describe the naïve point-in-polygon technique, followed by an improved algorithm which reduces the complexity of testing the point-in-polygon query by splitting complex polygons into smaller simple ones.

2.1.1 Naïve Point-in-Polygon Method

This approach [13] converts each pixel to a point and runs a traditional point-in-polygon test for it, as illustrated in Figure 2(a). First, it computes the minimum bounding rectangle (MBR) of the polygon. Then, it projects the MBR to the raster space, to find the corresponding range of pixels. After that, it scans all pixels in the projected MBR and for each pixel, it projects the pixel to a point in the vector space. After that, it tests if the point is inside the polygon. If the point is inside the polygon, it reads and processes the corresponding pixel value; otherwise, the pixel is skipped. The pixels found to be inside the polygon are grouped together and the desired aggregate function is run on them.

2.1.2 Quadratic-Split (QSplit) Method

The drawback of the *Naïve Point-in-Polygon Method* is that it has to test each pixel against the input polygons. For complex polygons which consist of hundreds of thousands of edges, a single point-in-polygon test can be costly. With very large raster datasets with trillions of pixels, the *Naïve Point-in-Polygon* approach is impractical.

To improve this approach, the Quadratic Split (QSplit) algorithm employs a spatial-join-inspired technique [9] which splits the polygon into four quadrants similar to the Quadtree splitting algorithm. Then, it recursively process each split, as shown in Figure 2. The final result will be the same because the center of each pixel will lie in exactly one of the four quadrants.

This algorithm is faster because running a point-in-polygon test for each of the smaller polygons can be much faster than testing the original polygon. Also, as shown in Figure 2, we can skip testing pixels in quadrants that are completely outside, and directly process all pixels in quadrants that are denoted as completely inside.

2.2 Raster-based Approaches

In this section, we describe the raster-based clipping and masking methods, which convert the vector dataset to raster to run the zonal statistics query.

2.2.1 Clipping Method

The clipping method [8], as shown in figure 3, uses the polygon to clip the raster layer by removing pixels that are outside the polygon. Since a raster layer is typically represented as a two-dimensional array, clipping is implemented by setting all *clipped* pixels to a special `NoData` value, e.g., -1. The clipped raster is represented as a new raster layer which is then processed using the desired aggregate function. The aggregate function skips all clipped pixels with the `NoData` marker and processes the values in other pixels. If the vector layer contains more than one polygon, each one is processed independently to generate one clipped raster per polygon. This method is implemented using PostGIS.

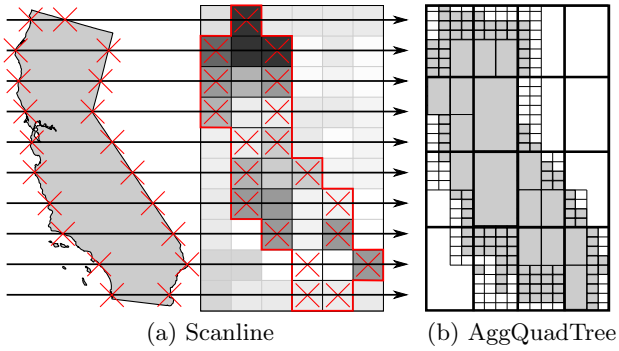


Figure 4: Raptor-based method

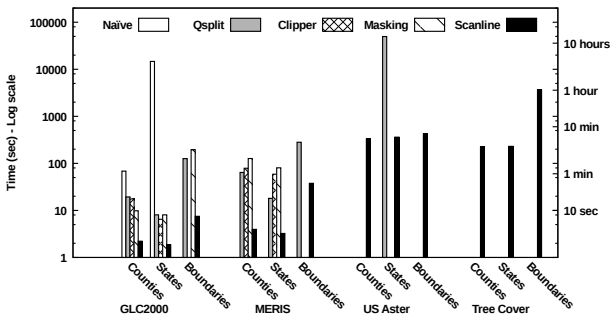


Figure 5: Performance experiments of all methods

2.2.2 *Masking Method*

This method [1] as illustrated in figure 3 runs in three steps, *rasterize*, *overlay*, and *grouped-aggregate*. The *rasterize* step, creates a *mask layer* for the input polygon where each pixel that lies inside the polygon is marked with the ID of the polygon, and pixels that are outside the polygon are marked with zero. The *overlay* step, combines the input raster layer with the mask layer to create one layer where each pixel has a pair of values, the pixel value from the input raster layer, and the polygon ID from the mask layer. The final *grouped-aggregate* step groups pixels by the polygon ID and computes the desired aggregate function on the pixel values, e.g., average. This method is implemented using SciDB.

2.3 Raptor-based Approaches

This section describes two raptor-based methods, *scanline* and *aggregate quad-tree*. Unlike the raster and vector-based methods, these method processes the two inputs in their raw format and do not require an explicit conversion from raster to vector or vice-versa.

2.3.1 *Scanline Method*

The scanline method [4] runs in three steps as shown in Figure 4. Step 1, calculates the Minimum Bounding Rectangle (MBR) of the input polygon(s) and maps its two corners to the raster layer to locate the range of rows to process in the raster layer.

Step 2 computes the intersections of each of the scanlines with the polygon boundaries.It converts each scan line to vector space and stores their y-coordinates in a sorted list. Each polygon is scanned for its corresponding range of scanlines, which are then used to compute intersections with the polygon. These intersections are then sorted by their x-coordinates for each scan line.

Step 3 finds the pixels that lie inside the polygons and process them. It maps the x-coordinates of the intersections to raster space and accumulates the corresponding pixel values. For multiple polygons, all intersections in one row are processed before moving to the next row.

This approach tries to reduce the intermediate storage for the intersection points. It also minimizes disk IO by scanning the raster data exactly once and by reading only the pixels that overlap the polygons. This method is IO-bound which makes it optimal for the processing perspective.

2.3.2 *Aggregate Quad-tree Method*

The scanline approach is efficient yet IO-bound. The aggregate quad-tree approach reduces the disk IO by providing partial aggregations on the raster data. These partial aggregates are stored in an aggregate quad-tree similar to the one used in [3] for rectangular queries.

Figure 4(b) portrays how the aggregate quad-tree method works. It first uses the scanline method to find all overlapping pixels that need to be processed. Then, based on the aggregate quad-tree structure, it fetches partial aggregates from aggregate quad-tree nodes that are *completely contained* in the query polygon. The method starts at the root node and tests if it is completely contained. If it is contained, its aggregate value is fetched and the search stops. Otherwise, its four child-nodes are recursively tested for containment. When the search reaches leaf nodes that are not completely contained, the underlying pixels from the original dataset are processed similar to the scanline method. All these values, i.e., partial aggregates and pixel values, are further combined together to produce the final result.

2.4 Discussion

Figure 5 compares the total running times of these methods. This experiment runs on vector datasets having upto 3 million segments and raster datasets having upto 800 billion pixels (782 GB). Each of the vector datasets comprises multiple polygons. More information about these datasets can be found in [4]. We omit running times for methods which failed to run or took too long. As shown in the Figure, the scanline method is able to scale to big raster and vector datasets while the other methods do not. Among the vector-based methods, QSplit methods performs better than the naïve method and scales to larger raster datasets. For the raster-based methods, both clipper and masking methods have almost the same performance and scalability.

3. DEMONSTRATION SCENARIO

During the demonstration, we will host the web application on a university web-server and make it accessible to the public audience. We will set up a laptop and a tablet for the audience to interact with the system and they can also access it from their smart devices. The web server will be pre-loaded with global raster datasets of temperature and vegetation over a period of 5 years, each having a different spatial resolution. The vector datasets loaded on the web server will include regional boundaries of countries, states, and cities (or their counterparts) all over the world.

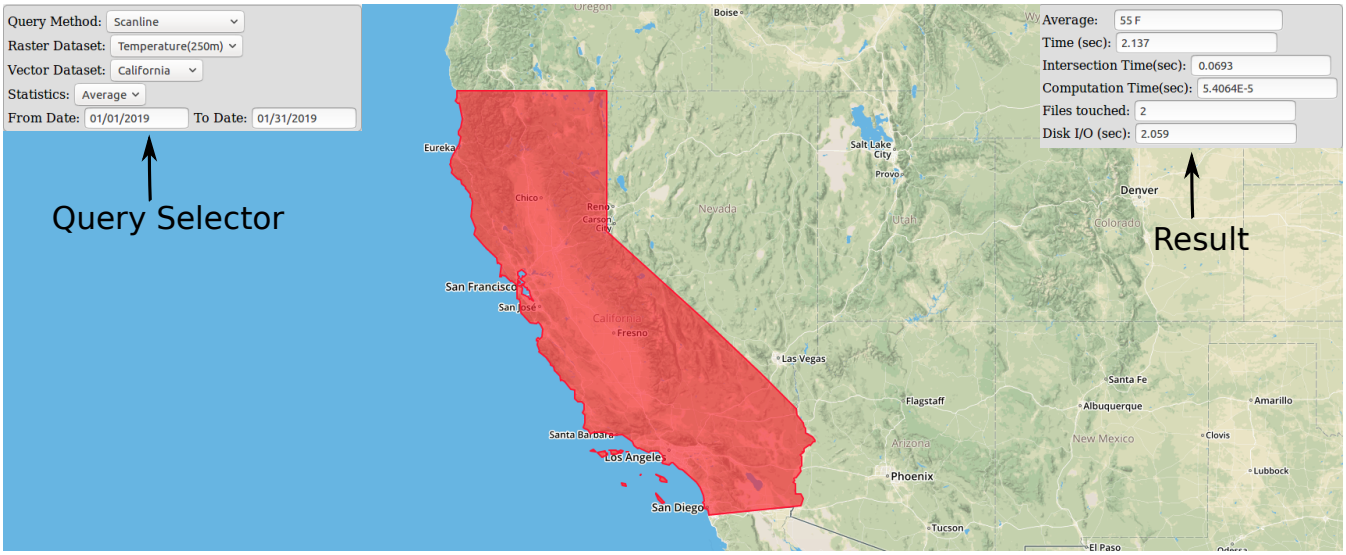


Figure 6: Raptor demonstration web interface

Web Interface: Figure 6 depicts the web interface which has selectors on the top-left to allow audience members to choose a query method, a raster dataset, a temporal range for the raster dataset, a vector dataset and the required statistics. The query results will be shown on the top-right along with a few performance metrics. The performance metrics will include the total running time for the query as well as a breakdown of the time in different steps, number of files accessed, number of pixels accessed, and the disk I/O.

The attendees can run queries like: *"Which state in my country recorded the highest average temperature this year?"*, using either of the techniques. It will allow them to compare and observe the variations among the different techniques, using the performance metrics. They can also run this query using two different resolutions of temperature dataset and observe the effect of increased resolution on the running times for the techniques. Queries involving different polygons, such as region boundaries for a big state, e.g., *California*, or a small state, e.g., *Wyoming*, will help the attendees observe the advantage of the QSplit method over the naïve point-in-polygon method. California makes a very complex polygon as compared to Wyoming, hence making the point-in-polygon test very expensive to run. Since, QSplit method deals with complex polygons, it will show a performance gain for California while there will not be a major difference in performance for Wyoming.

The attendees can also run queries like: *"Calculate the average rainfall for each county in my country."* using either of the techniques. They may also choose any other statistic like maximum or minimum, or may choose to see these statistics for each year over a range of years, and for different regional levels in the world.

4. FUTURE WORK

This demo provides a comparison of the three approaches, namely, raster, vector and raptor for the zonal statistics problem. The raptor approach provides an efficient method that processes raster and vector data in their native forms. In future, we plan to show that this approach can be easily adapted for other raptor queries like areal interpolation.

Acknowledgment

This work is supported in part by the National Science Foundation (NSF) under grant IIS-1838222 and by the USDA National Institute of Food and Agriculture, AFRI project 2018-07212.

5. REFERENCES

[1] Zonal Statistics in ArcGIS. http://bit.ly/arcgiszs, 2017.
[2] P. Baumann et al. The multidimensional database system rasdaman. In *SIGMOD*, pages 575–577, 1998.
[3] A. Eldawy et al. SHAHED: A MapReduce-based System for Querying and Visualizing Spatio-temporal Satellite Data. In *ICDE*, pages 1585–1596, Seoul, Korea, Apr. 2015.
[4] A. Eldawy et al. Large scale analytics of vector+raster big spatial data. In *SIGSPATIAL*, pages 62:1–62:4, 2017.
[5] A. Eldawy and M. F. Mokbel. SpatialHadoop: A MapReduce Framework for Spatial Data. In *ICDE*, 2015.
[6] D. Haynes et al. Terra Populus' Architecture for Integrated Big Gepspatial Services. *Transactions on GIS*, 2017.
[7] G. D. Jenerette et al. Ecosystem Services and Urban Heat Riskscape Moderation: Water, Green Spaces, and Social Inequality in Phoenix, USA. *Ecological Applications*, 2011.
[8] A. Kini and R. Emanuele. Geotrellis: Adding Geospatial Capabilities to Spark, 2014.
[9] S. Ray et al. Skew-resistant Parallel In-memory Spatial Join. In *SSDBM*, pages 6:1–6:12, 2014.
[10] M. Stonebraker et al. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science and Engineering*, 15(3):54–62, 2013.
[11] D. Xie et al. Simba: Efficient In-Memory Spatial Analytics. In *SIGMOD*, 2016.
[12] J. Yu et al. GeoSpark: A Cluster Computing Framework for Processing Large-Scale Spatial Data. In *SIGSPATIAL*, pages 70:1–70:4, 2015.
[13] J. Zhang, S. You, and L. Gruenwald. Efficient Parallel Zonal Statistics on Large-Scale Global Biodiversity Data on GPUs. In *BIGSPATIAL*, pages 35–44, 2015.