Distributed Zonal Statistics of Big Raster and Vector Data*

Samriddhi Singla
Computer Science and Engineering
University of California, Riverside
ssing068@ucr.com

Ahmed Eldawy
Computer Science and Engineering
University of California, Riverside
eldawy@ucr.com

ABSTRACT

The recent advances in remote sensing technology resulted in peta bytes of data in raster format. To process this data, it is often combined with high resolution vector data that represents, for example, region boundaries. One of the common operations that combine big vector and raster data is the zonal statistics which computes some aggregate values for each polygon in the vector dataset. This paper proposes a novel and scalable algorithm for zonal statistics that can scale to peta bytes of raster and vector data. The proposed method does not require any preprocessing or indexing making it perfect for ad-hoc queries that scientists usually want to run. We implement a prototype for the proposed method and the initial preliminary results show that the proposed method can scale up-to a trillion pixels.

CCS CONCEPTS

• **Information systems** → *Database query processing*;

KEYWORDS

Big Spatial Data, Raster, Vector, Satellite, Clipping, Zonal Statistics

ACM Reference format:
Samriddhi Singla and Ahmed Eldawy. 2018. Distributed Zonal Statistics of Big Raster and Vector Data. In *Proceedings of 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, November 6–9, 2018 (SIGSPATIAL '18),* 4 pages.
https://doi.org/10.1145/3274895.3274985

1 INTRODUCTION

Remote Sensing Data is of vital importance to various research domains, such as, agriculture, environmental studies, and oceanography. It has been used to study climate change, map land and vegetation change and has numerous other applications. Recently, there has been a tremendous increase in the amount of this data with the advancements in remote sensing technology. NASA EOSDIS provides public access to more than 17 petabytes of Earth Observational data, which is estimated to grow to more than 330 petabytes by 2025 [3]. European Space Agency(ESA) has collected over five petabytes of data within two years of the launch of the Sentinel-1A satellite and is expected to receive data continuously until 2030 [4].

The remote sensing data is in the raster format, and its use requires it to be often processed in combination with vector data. Zonal Statistics is a fundamental operation which requires to process the combination of raster and vector data to compute aggregate values for a zone defined by the vector data using the values provided by the raster data. It is used in many applications, including the study by ecologists on the effect of vegetation and temperature on human settlement [7, 8] and by geographers for analyzing terabytes of socio-economic and environmental data [5, 6].

To make use of the ever-growing amount of spatial data, there is a need of scalable distributed techniques that can efficiently process it. The existing systems for big spatial data include SpatialHadoop, GeoSpark, Simba, SciDB [10], RasDaMan [1], and GeoTrellis [9]. The above mentioned systems are very efficient, however, they focus on either processing big raster data or big vector data, and provide a poor performance when the combination of vector and raster data needs to be processed.

Traditional methods to process the zonal statistics problem focused on either vectorizing the raster dataset [11] or rasterizing the vector data [6]. Both suffer from the drawback of running a costly conversion process which makes them unscalable to high-resolution raster and vector data. To overcome this drawback, the ScanLine [2] method was proposed recently which processes the two datasets in their native format without a need for a conversion process. It proved to be very efficient in producing the best performance on a single machine but it was still limited to the resources available on a single machine.

In this paper, we study the problem of distributed zonal statistics on high-resolution raster and vector data. The basic ScanLine method relies on loading the entire vector layer in memory and hence cannot scale to large vector layers. To overcome the above limitation, we propose an efficient MapReduce implementation (EMI) for the zonal statistics problem. The algorithm runs in two phases, namely, preparation and aggregation phases. The preparation phase runs on a single machine and efficiently performs the common logic that is needed by all the machines. At the same time, it gets the chance to look into the metadata of the raster and vector files and decide how to efficiently split the job across machines. The second aggregation phase is then launched as a MapReduce job that scans the relevant parts of the raster files and computes the desired aggregate values efficiently.

In this paper, we implement a prototype of the proposed idea and run some preliminary experiments to study its applicability. The results are promising and show that the proposed idea can overcome some of the limitations of the state-of-the-art ScanLine method. The experiments also reveal a new bottleneck in the intersection computation that we plan to address in the future.

*This work is supported in part by the National Science Foundation (NSF) under grant IIS-1838222.

SIGSPATIAL '18, November 6–9, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-5889-7/18/11…$15.00
https://doi.org/10.1145/3274895.3274985

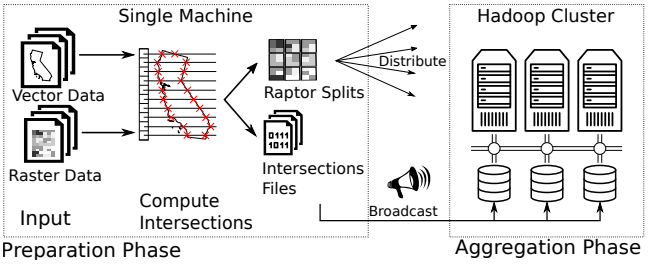

Figure 1: System Overview

2 DISTRIBUTED ZONAL STATISTICS

The input to the zonal statistics problem is a raster layer, a vector layer that comprises a set of polygons, and an aggregate function. The output is the value of the aggregate function when applied to all pixels that overlap with each polygon separately. For example, this operation can compute the average temperature of each state in the US on a specific date. In this case, the raster layer represents the temperature values on the selected date, the vector layer represents the polygon boundaries of the US states, and the aggregate function is the average.

The state-of-the-art ScanLine method [2] runs by first calculating all the intersections between the polygons boundaries in the vector layer and the centers of the scan lines in the raster layer. Then, it uses these intersections to scan over the raster layer exactly once while incrementally computing the aggregate functions for all the polygons in the vector layer. This method proved to be very efficient in minimizing the memory footprint and disk IO. However, it had a major limitation of running on a single machine which makes it limited to the capabilities of that machine. This section describes the proposed distributed algorithm for zonal statistics.

2.1 Overview

The proposed Efficient MapReduce Implementation (EMI) takes a novel approach to parallelize the ScanLine method. Figure 1 provides an overview of the proposed algorithm. While analyzing the baseline method, we observed that the bottleneck is in the IO cost of the aggregation phase that reads the big raster data. Therefore, as a first step for parallelizing this method, we decided to keep the first phase (intersection computation) as a single machine process while only distributing the aggregation phase over multiple machines using MapReduce. As shown in Figure 1, the proposed algorithm runs in two phases, namely, a single-machine *preparation phase* and a distributed MapReduce *aggregation phase* both outlined below.

- *Preparation Phase*: As illustrated in Figure 1, this phase runs on a single machine and performs two tasks. First, it computes the intersections of the geometries in the vector file with the raster layer similar to the ScanLine method. These intersections are written to binary files called the *Intersection files*, which are broadcast to all the machines. Second, it defines a logical partitioning for the raster and vector files and creates RAster Plus vecTOR splits, termed *Raptor Splits*, which define the smallest units of work and are then distributed to the machines during the second phase.
- *Aggregation Phase*: This phase runs as a MapReduce job, where each mapper receives a *Raptor Split* that defines a subset of raster tiles and polygons to process. The map function reads the specified tiles and the intersection file that corresponds to the polygons and computes a partial answer for the zonal statistics. The partial answers are combined by polygon ID and the reducers combine them to produce the final aggregate.

2.2 Preparation Phase

The goal of this phase is to prepare and create the MapReduce job that computes the zonal statistics. It runs on a single machine on the head node of the cluster and performs two tasks, namely, *intersection file generation* and *raptor split generation*. The intersection file generation step computes a common structure, called *intersection file*, which is broadcast to all the machines to be used in the second distributed *aggregation phase*. The raptor split generation step creates a list of tasks that is distributed among machines to perform the parallel computation. Below, we describe the two tasks in more details.

Intersection File Generation

From our previous work, we showed that the intersection computation part requires a minimal overhead as compared to statistics computation phase. Therefore, running it once on a single machine is more efficient than running it thousands of times on multiple machines. To compute the intersections, a chunk of the vector file is loaded into memory and only the metadata of the raster file is loaded, e.g., resolution and coordinate reference system (CRS). For each chunk of polygons, we run the first phase of the scan line algorithm which computes the intersections between the polygons and each row of pixels in the raster layer. For each row, the intersections are represented as a list of pairs $\langle x, pid \rangle$ sorted by x, where x is the coordinate of an intersection and pid is the ID of the polygon intersecting at that position. All these intersections are then written to a compact binary file called the *intersection file*. If multiple raster files are given in the input, this step uses multithreading to compute the intersections with each raster file in parallel. For efficiency, each thread writes its part of the intersection file to an *interim file* independently and they are finally concatenated in one intersection file. Each intersection file has a footer which stores the range of polygon IDs covered by the corresponding chunk of vector file and a list of offsets in the file for the sections in that file, one for each raster file. Writing the interim files immediately is crucial to reduce the memory overhead as both the polygons and computed intersections can be evicted from memory right after. Once all threads finish, one thread concatenates the sections that correspond to one file and adds the footer to it. The concatenation step does not add a huge overhead and it makes the organization of these files easier.

Raptor Split Generation

The second task, *Raptor Split Generation*, performed by this phase, generates *Raptor Splits* using the *RaptorInputFormat*. In Hadoop, the InputFormat is the component that splits the input file into equi-sized splits to be distributed on the worker nodes. These splits are mapped one-to-one to mappers. Therefore, each split defines a unit of work. A corresponding *record reader* uses the split to extract key-value pairs that are sent to the map function for processing. Since

our unit of work is a combination of raster plus vector data, we define the new *RaptorInputFormat*, *RaptorSplit*, *RaptorRecordReader*, and *RaptorObject*. Starting with the smallest one, the RaptorObject contains vector chunk ID, a raster file ID, and a tile ID in that raster file. In the next phase, the map function processes one RaptorObject at a time. The RaptorSplit stores a vector chunk ID, a raster file ID, and a *range* of tile IDs in that raster file. The RaptorSplit defines a unit of work given to a mapper. We can control the amount of work given to each mapper by adjusting the number of tiles in the range. The RaptorRecordReader takes one RaptorSplit and iterates over all the RaptorObjects that it represents. Finally, the RaptorInputFormat takes all the input to the problem, i.e., the raster files and all intersection file, and produces a list of RaptorSplits that define the map tasks given to worker nodes. Notice that the preparation phase only deals with the RaptorInputFormat and generates a list of RaptorSplits out of the input. Therefore, this single-machine step is extremely fast as it does not involve any processing of either vector or raster data.

The information needed to logically partition the raster file is generated when the intersection of a raster file with a *vector chunk* is computed. The definition of the tiles is part of the metadata of the raster file which is loaded to compute the intersections. Moreover, for efficiency, while computing the intersections, we keep track of the tiles that actually overlap the polygons and we make sure to generate RaptorSplits that cover only those tiles. In other words, this step prunes all the tiles that do not contribute to the answer.

The number of generated *RaptorSplits* depends on the total number of Intersection files, raster tiles, and number of tiles. For efficiency, each RaptorSplit is limited to one vector chunk and one raster file. This ensures that each mapper will need to load exactly one section of the intersection file and open one raster file only.

2.3 Aggregation Phase

This phase is implemented as one MapReduce job, where each machine runs a map function to compute partial Zonal Statistics for the *RaptorSplit* assigned to it. The *RaptorSplit* is a set of RaptorObjects, where each object contains a vector chunk ID, a rater file ID, and a tile ID. The mapper starts by reading the section of the intersection file identified by the chunk ID and raster file ID. A copy of it is cached to process future tiles in the same raster file. Then, the mapper processes the tile identified by the tile ID by loading and aggregating the pixels identified by the ranges in the intersection file. The output is a set of pairs $\langle p_i, a_i \rangle$, where p_i is the polygon ID and a_i is the statistics computed for p_i in the given tile. The reduce function merges the partial statistics a_i belonging to the same polygon p_i and outputs the final aggregation $\sum a_i$.

2.4 Hadoop Vs Spark

Even though the Efficient MapReduce Implementation could have been implemented using both Spark and Hadoop, we chose to do it in Hadoop. The strength of the proposed implementation lies in parallelizing the costly I/O aggregation step. While Spark can be slightly faster by improving the computation, Hadoop is expected to be on par with it as it follows the same logic.

Table 1: Vector and Raster Datasets

Vector datasets

Dataset	Polygons	Segments	$\frac{\#segments}{\#polygons}$	File Size
Counties	3,108	51,638	17	978 KB
States	49	165,186	3,370	2.6 MB
Boundaries	284	3,817,412	13,440	60 MB
TRACT	74,133	38,467,094	519	632 MB
ZCTA5	33,144	52,894,188	1596	851 MB

Raster datasets

Dataset	Resolution	File Size
glc2000	40,320×16,353	629 MB
MERIS	129,600×64,800	7.8 GB
US-Aster	208,136×89,662	35 GB
Tree cover	1,296,036×648,018	782 GB

3 PRELIMINARY RESULTS

This section provides an experimental evaluation of the Efficient MapReduce Implementation (EMI) as compared to the Scanline Method using real data. Section 3.1 describes the setup of the experiments and the datasets used, while Section 3.2 provides the results.

3.1 Setup

We run all the experiments on a cluster with one head node and 12 worker nodes. The head node has Intel(R) Xeon(R) CPU $E5-2609$ v4 @ 1.70GHz processor, 128 GB of RAM, and 2x8-core processors running CentOS and Oracle Java 1.8.0_131. The worker nodes have Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz processor, 64 GB of RAM, and 2x6-core processors running CentOS and Oracle Java 1.6.0_31-b04. The methods are implemented using the open source Geotools library 17.0.

In all the techniques, the aggregate function computes the four aggregate values, minimum, maximum, sum, and count and perform comparison based on end-to-end running time. Table 1 lists the datasets that are used in the experiments. The vector layers represent the US continental counties and US continental states with 3,000 and 49 features respectively. The Large-Scale International Boundaries (LSIB) includes geographic national boundaries for 249 countries and disputed areas. The TRACT and ZCTA5 dataset are a part of TIGER 2017 dataset. The raster datasets come from various government agencies. The GLC2000 and MERIS 2005 datasets are from the European Space Agency with pixel resolutions of 0.0089 decimal degrees (1km) 0.0027 (300m) respectively. The US Aster dataset originates from the Shuttle Radar Topography Mission (SRTM) and covers the continental US. Hansen developed the global Tree Cover change dataset which covers the entire globe. Both datasets have a spatial resolution of 0.00028 decimal degrees (30m).

3.2 Overall Comparison

Figure 2 provides a comparison of the Efficient MapReduce Implementation (EMI) and the Scanline Method [2]. It can be observed that the Efficient MapReduce Implementation is scalable for larger

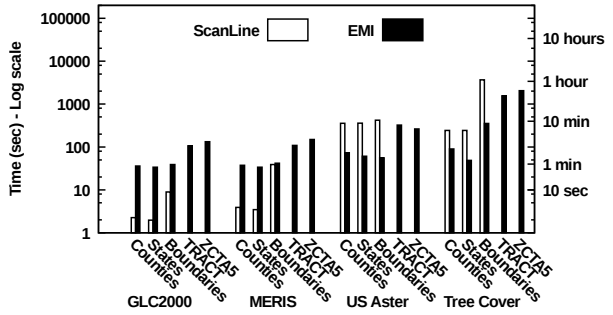

Figure 2: Overall comparison of ScanLine and EMR algorithms for different vector and raster datasets

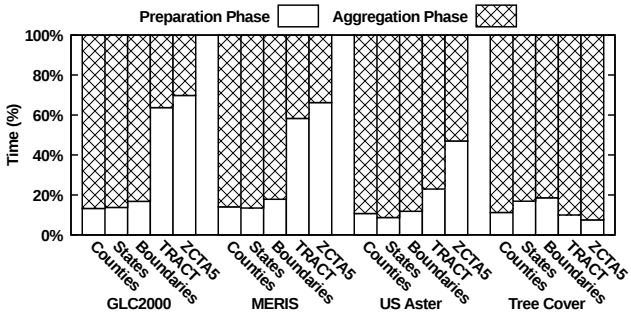

Figure 3: Breakdown of running time for EMI

vector datasets (TRACT and ZCTA5), while the single-machine ScanLine Method fails (runs out of memory) for them. For large raster datasets, the proposed EMI algorithm is much faster than the Scanline algorithms with up-to an order of magnitude speedup. There is an expected decrease in performance for smaller raster datasets due to the additional overhead incurred in setting up a MapReduce job in Hadoop. Its performance is at par to ScanLine method for the MERIS dataset and Boundaries dataset, and then increases as size of one of the raster or vector datasets increases.

The scalability of the proposed algorithm can be attributed to decision of creating vector chunks and *interim files*. Its gain in performance over single-machine ScanLine Method for larger datasets is due to the distributed computation of Zonal Statistics.

3.3 Breakdown of Total Running Time

Figure 3 shows the breakdown of the total running time for EMI into two phases, preparation phase and aggregation phase. All the numbers shown in Figure 3 are normalized to the overall running time for comparison. The actual numbers are same as in Figure 2.

This experiment reveals a new bottle neck in the preparation phase that only appeared after parallelizing the aggregation phase. As shown in the figure, the running time is dominated by the *preparation phase* for the combination of smaller raster datasets (GLC200 and MERIS) with the larger vector datasets (TRACT and ZCTA5). For all other combinations of raster and vector datasets, the running time is dominated by the *aggregation phase*. The reason for the domination of preparation phase for GLC200 and MERIS against TRACT and ZCTA5 datasets is because of the large number of geometries in the vector files for whom the intersections with the raster file must be computed. Also, the small size of raster files lead to a small number of logical partitions for which the zonal statistics must be computed, hence the aggregation phase takes less time than the preparation phase, for these datasets. Moreover, the computation of Zonal Statistics require reading the required pixel values from disk for each *Raptor Split*. This makes the aggregation phase dominated by disk IO for reading only the required pixel values. This leads to the running time for aggregation phase becoming dominant over that for preparation phase for large raster files.

4 CONCLUSION AND FUTURE WORK

In this paper, we presented a distributed MapReduce algorithm for the zonal statistics problem. The proposed algorithm provides key ideas that can carry on to other distributed algorithms for processing big vector and raster datasets. It runs in two phases, a single-machine preprocessing step that computes a common data structure to be used in parallel, and defines the tasks that will be executed in parallel. The second phase runs in parallel and aims at reading and processing the big raster files efficiently. Our experiments show that the proposed algorithm can scale to very large data whereas the baselines could not handle big vector or raster data. We intend to study in future the effect of *RaptorInputFormat* and creation and compression of intersection files on the proposed algorithm. We also intend on exploring the parallelization of preparation phase and using indexing techniques to spatially partition vector data and(or) raster data, to provide a better distribution of work among machines.

REFERENCES

[1] Peter Baumann and others. 1998. The Multidimensional Database System RasDaMan. In *SIGMOD*. Seattle, WA, 575–577.
[2] Ahmed Eldawy, Lyuye Niu, David Haynes, and Zhiba Su. 2017. Large Scale Analytics of Vector+ Raster Big Spatial Data. (2017).
[3] EOSDIS 2017. The Common Metadata Repository: The Foundation of NASA's Earth Observation Data. (2017). https://earthdata.nasa.gov/the-common-metadata-repository.
[4] ESA 2017. The ESA Earth Observation Payload Data Long Term Storage Activities. (2017). https://www.cosmos.esa.int/documents/946106/991257/13_Pinna-Ferrante_ESALongTermStorageActivities.pdf/813babe0-58db-4e23-b710-3bd9d6b58b12.
[5] David Haynes and others. 2015. High Performance Analysis of Big Spatial Data. In *Big Data*. Santa Clara, CA, 1953–1957.
[6] David Haynes, Steven Manson, and Eric Shook. 2017. Terra Populus' Architecture for Integrated Big Gepspatial Services. *Transactions on GIS* (2017).
[7] G. Darrel Jenerette and others. 2011. Ecosystem Services and Urban Heat Riskscape Moderation: Water, Green Spaces, and Social Inequality in Phoenix, USA. *Ecological Applications* 21 (2011), 2637–2651. Issue 7.
[8] G Darrel Jenerette, Sharon L Harlan, Anthony Brazel, Nancy Jones, Larissa Larsen, and William L Stefanov. 2007. Regional Relationships Between Surface Temperature, Vegetation, and Human Settlement in a Rapidly Urbanizing Ecosystem. *Landscape Ecology* 22 (2007), 353–365. Issue 3.
[9] Ameet Kini and Rob Emanuele. 2014. Geotrellis: Adding Geospatial Capabilities to Spark. (2014).
[10] Michael Stonebraker, Paul Brown, Donghui Zhang, and Jacek Becla. 2013. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science and Engineering* 15, 3 (2013), 54–62.
[11] Jianting Zhang, Simin You, and Le Gruenwald. 2015. Efficient Parallel Zonal Statistics on Large-Scale Global Biodiversity Data on GPUs. In *BIGSPATIAL*. Bellevue, WA, 35–44.