SATO: A Spatial Data Partitioning Framework for Scalable Query Processing

Hoang Vo
Emory University
hvo8@emory.edu

Ablimit Aji*
HP Labs
ablimit@hp.com

Fusheng Wang
Emory University
fusheng.wang@emory.edu

ABSTRACT

Scalable spatial query processing relies on effective spatial data partitioning for query parallelization, data pruning, and load balancing. These are often challenged by the intrinsic characteristics of spatial data, such as high skew in data distribution and high complexity of irregular multi-dimensional objects. In this demo, we present SATO, a spatial data partitioning framework that can quickly analyze and partition spatial data with an optimal spatial partitioning strategy for scalable query processing. SATO works in following steps: 1) **S**ample, which samples a small fraction of input data for analysis, 2) **A**nalyze, which quickly analyzes sampled data to find an optimal partition strategy, 3) **T**ear, which provides data skew aware partitioning and supports MapReduce based scalable partitioning, and 4) **O**ptimize, which collects succinct partition statistics for potential query optimization. SATO also provides multiple level partitioning, which can be used to significantly improve window based queries in cloud based spatial query processing systems. SATO comes with a visualization component that provides heat maps and histograms for qualitative evaluation. SATO has been implemented within the Hadoop-GIS, a high performance spatial data warehousing system over MapReduce. SATO is also released as an independent software package to support various scalable spatial query processing systems. Our experiments have demonstrated that SATO can generate much balanced partitioning that can significantly improve spatial query performance with MapReduce comparing to traditional spatial partitioning approaches.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial Database and GIS, Scientific databases*

Keywords

Database, Data Warehouse, Spatial Partitioning, MapReduce, Scientific Data Management, Visualization

1. INTRODUCTION

The proliferation of embedded devices and rapid advancement in sensor technology have enabled enterprises to collect massive amounts of spatial data that provide information about the geographical locations and trajectories of objects of interest. Timely analysis and management of such spatial data is not only essential to business growth and better location intelligence for enterprises, but also a basic infrastructure requirement for the realization of Internet of Things paradigm. The massive amounts of data, coupled with the need for complex spatial analytics, require a high-throughput and low latency query processing approach.

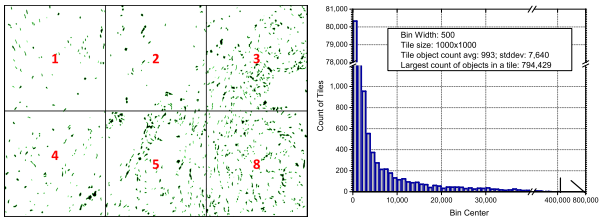
Data partitioning is a powerful mechanism for improving efficiency of data management systems, and it is a standard feature in modern database systems. For example, as of this writing, all major DBMS vendors have horizontal and vertical partitioning techniques built into their system. Aside from the fact that data partitioning improves the overall manageability of large datasets, it improves query throughput and latency in two ways. First, partitioning the data into smaller units enables processing of a query in parallel, and henceforth the improved throughput. Second, if the partitioning is performed effectively, I/O can be significantly reduced by only scanning a few partitions that contain relevant data. For example, consider a simple spatial query *find all tweets that were tweeted within a mall in California*. If the tweets are spatially partitioned by their geographical location at the state boundary level, only the partitions that contain California related tweets are need to be scanned. However, if the dataset is not spatially partitioned, a brute-force approach essentially performs a rather expensive whole table scan operation.

Spatial data partitioning is a less explored research topic compared to their relational counterpart, while being critical for query performance. To the best of our knowledge, no spatial database system provides a graceful approach to partition on the spatial attribute. Previously, Paradise [7] – a parallel spatial database system, used a *regular fixed grid* partitioning for parallel join processing. However, there are several problems with this approach as described in the original work. i) As spatial objects (e.g. polygons and polylines) have extent, regular grid based spatial partitioning would undesirably produce objects spanning multiple cell grids, which need to be replicated and post processed. If such objects account for a considerable fraction of the dataset, the overall query performance would suffer from such boundary handling overhead. ii) Fixed grid partitioning is skew averse, whereas data in most real world spatial applications are inherently highly skewed. For example, in OpenStreetMap (OSM), certain regions have more data compared to others due to enthusiastic data contributors. If OSM is partitioned with fixed grid approach with 1000x1000 grids, the average count of objects per tile is 993, but the tile with most objects contains $794,429$ objects. In such case, it is very likely that par-

*work was done as a Ph.D. student at Emory University.

SIGSPATIAL'14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA
ACM 978-1-4503-3131-9/14/11 ...$15.00
http://dx.doi.org/10.1145/2666310.2666496 .

allel processing nodes assigned to process those dense regions will become the *stragglers*, and the overall query processing efficiency will be much affected. [9].



(a) Fixed grid partition (b) Spatial data skew

Figure 1: An example of fixed grid partition and spatial data-skew

In this paper, we present SATO, an effective and scalable partitioning framework that can partition a geospatial dataset into *balanced regions* while *minimizing the number of boundary objects*. The partitioning methods are designed for scalability, which can be easily parallelized for high performance, for example, running in MapReduce. SATO stands for four main steps in this framework for spatial data partitioning: **S**ample, **A**analyze, **T**ear, and **O**ptimize. First, we sample a small fraction of the dataset to identify overall global data distribution with potential dense regions. Next we analyze the sampled data with a *partition analyzer* that produces a coarse partition scheme in which each partition region is expected to contain roughly equal amounts of spatial objects. Then we pass these coarse partition regions into the partitioning component that *tears* the regions into more granular partitions that are data skew aware and meet partition requirements. Finally, we analyze the generated partitions to produce multi-level partition indexes and additional partition statistics which can be used to *optimize* queries. SATO is also implemented for parallelization. Besides running as a standalone program, it is also integrated with Hadoop-GIS, a scalable MapReduce based spatial query processing framework [2]. In this demo, we use a MapReduce based query processing model to demonstrate that how effective partitioning can improve query performance. However, the approaches described here can be completely applicable for any other distributed spatial data management systems [3, 5, 8] that utilize partitioning for query processing and optimization.

The rest of the demonstration proposal is organized as follows. In Section 2 we describe the SATO framework in details. In Section 3, we introduce how SATO parallelizes the partitioning process and how the results of the partitioning are used to facilitate query processing. In Section 4, we report performance results. In Section 5, we illustrate the demonstration details and how the system can be used by a typical user. Finally, we conclude the paper in Section 6.

2. THE SATO FRAMEWORK

SATO assumes an input dataset contains a spatial field in the WKT (Well-known text) format – other data representations can be easily included and extended. The data type of the spatial field can be any of Open Geospatial Consortium-support (OGC) spatial data types. The dataset can contain other feature fields or attributes. If a feature or attribure is provided, it is considered to be associated with the corresponding spatial object, and co-located with the spatial partition which contains the spatial object. Next, we explain the four key steps in SATO.

2.1 Sample

In relational database systems, sampling is used in various tasks to avoid full dataset scan. For example, typical histogram construction algorithms work on a small fraction of sampled data, thus avoid the expensive computation of full dataset statistics. In the sampling step, we apply a stratified sampling approach to sample the dataset. One important control parameter here is the sampling ratio. Our experiments show that a fraction of 1~3% is enough to fairly capture the density distribution of datasets. Depending on the domain of the dataset or the potential query workload, the sampling approach can be tweaked to satisfy domain constraints. For example, if users frequently run a join query between road objects and river objects, we may statistically increase the sampling ratios of those objects, so that the resulting partitioning scheme is optimized for the join query.

2.2 Analyze

We use the Minimum Bounding Rectangle (MBR) of spatial objects in the sampled dataset as the approximation of their spatial extent, and feed them into the analyzer. The analyzer analyzes spots of high density, and the overall data distribution to derive an optimal *global* partitioning scheme, which potentially produce fewer boundary objects, while ensuring that the generated partitions are balanced at the global partition level. Furthermore, for batch processing systems such as Hadoop, we tweak the analyzer to generate larger partitions to match the HDFS file block size. Partitions of size similar to the HDFS file block size avoids file fragmentation and improves I/O efficiency of future queries.

We have built a collection of six partition algorithms integrated in the analyzer. We omit the technical details of those algorithms which are described in the full paper. For each partitioning task, the analyzer will assign one of the six algorithms that produces the best partitioning result. For example, for global partitioning, analyzer runs on the sampled dataset to produce a global partitioning; then for each global partition, the analyzer runs on that specific partition to derive an optimal partitioning scheme used to further refine that partition into smaller ones.

2.3 Tear

In this step, we further partition each of the global partitions into smaller *local* partitions. For each global partition, we use the partitioning algorithm, which was designated in the analyze step, to further refine the partition. The actual partitioning of the data is executed in this step, and the output for each spatial object contains a partition id that maps to a corresponding partition boundary. In our prototype implementation, the local partition boundary is encoded to the partition id, so that it may be further utilized by certain query processing tasks. Since each partition task is independent of the other, the partitioning can be done in parallel. In the demonstration, we use a MapReduce based parallelization to match the query processing model of Hadoop-GIS. However, it is not challenging to implement such partitioning in any other high performance computing model.

In this step, we can also instruct the partitioner to generate partitions of specific size. For example, for parallel spatial database systems, we can generate partitions that match the DBMS page size which can increase the data loading performance and query processing performance.

2.4 Optimize

After the partitioning is done, we optionally re-scan the data to perform two complementary optimization tasks. First, for each partition we collect basic statistics, such as the number of objects, the geometry of partition boundary, and the number of boundary ob-

jects. The partitions statistics is persisted to the catalog for future use, and a multi-level partition index is constructed from the partitions boundaries and persisted to the file system. This multi-level index contains a mapping between file block information and partition boundary which can be later utilized by the query processing task. Figure 2 shows a simplified version of a multi-level index for MapReduce based spatial query processing systems. Second, we re-evaluate each partition to ensure that the generated partitions are balanced. As the partitioning is performed on the sampled dataset, it is possible that some *oversized* or *thin* partitions are produced due to over/under sampling. If we find such a partition, we start a repartition process, which further partitions the oversized partition into multiple smaller ones to meet the partition size constraint. In contrast, for thin partitions, we merge these with neighboring partitions without violating the partition size constraint.

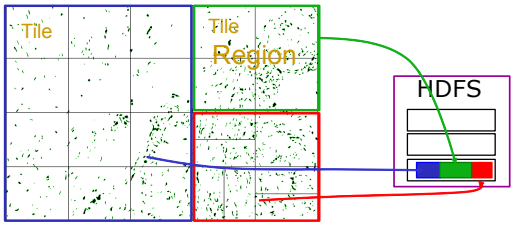


Figure 2: Multi-Level region based spatial index

3. PARTITIONING AND PARALLELIZATION

3.1 Partitioning Approaches in SATO

SATO has six built in partitioning algorithm that can handle various datasets from different domains, and each of them has its own merits. Here, we sketch out those algorithms, and interested readers can find specifics from the full technical paper. During the demonstration, we will visually present why certain algorithms are preferred for a specific partition task by illustrating the advantages and disadvantages of those approaches.

Fixed Grid Partitioning. As the name indicates, the spatial universe is partitioned into K equal sized grids, and each grid is a single partition.

Binary Split Partitioning. This is a top-down approach that creates spatial partitions by recursively dividing a given spatial region into two non-overlapping subregions until the partitioning satisfies the page size constraint.

Hilbert Curve Partitioning. This is commonly in many applications to obtain an approximate total ordering which preserves spatial locality for multidimensional data. In our implementation, we used Hilbert curve to map the centroids of the spatial objects to obtain ordering values, and the whole dataset is sorted based on the values. Then each consecutive b objects are grouped together to satisfy the expected page size constraint, and their spatial extents are used to form a spatial partition.

Strip Partitioning. In this approach, rather than defining a fixed space, we slice off an "appropriate" amount of space from the spatial universe where each slice roughly contains c objects.

Boundary Optimized Strip Partitioning. This algorithm is designed to reduce the number of boundary objects while partitioning. It similar to Strip partitioning with the extension that it makes greedy choice when producing partitions. While performing the strip based partitioning, we consider partitioning in both dimensions, and select the one which induces a smaller number of boundary objects.

Sort-Tile-Recursive (STR) Partitioning. STR [4] first partitions the spatial universe into large vertical strips. Then each strip is further partitioned in the horizontal direction.

3.2 MapReduce Based Parallelization

In large scale query systems, partitioning is a one time process, and the cost of partitioning is amortized by the improved query performance in the long run. However, there are certain queries that can benefit from an immediate and efficient repartition. Moreover, in practice, parallel partitioning is considered to be one effective way to improve overall ETL efficiency in large scale systems. If the spatial partition process is a single-thread program, the memory and computation time might be intractable for very large datasets.

We propose and implement two new scalable MapReduce–based spatial partitioning approaches for this demonstraion. Both of these approaches are used in the *Analyze* step to generate partition regions. The first approach is designed for the Hilbert Curve where we use a technique that resembles Hadoop Terasort [6] to sample data for total ordering, and sorts values based on such partitioning. In the Map phase of MapReduce, a spatial ordering anchor, such as the center or Hilbert Curve value, is calculated and emitted as the key. Next, the MapReduce framework will partition the objects into groups based on their anchor location and sorts them on the anchor value. At this point, dataset has been partitioned at a very coarse granularity. Next, when the reduce phase starts, each reducer will work on a single coarse partition, and further partitions them into smaller partitions. Thus, at the end of the reduce phase, we have the final partitioning layout. The second approach is more universal and applicable to all algorithms. We perform an extremely fast spatial histogram construction, and divide the space into large regions with roughly equal numbers of objects. Our specific implementation uses a preliminary spatial density estimation, and successively divides the largest rectangular region along one specific dimension producing the minimal skew similar to the *Min-skew* approach in [1]. The advantages of this approach is that the histogram computation on input data chunks can be performed independently and quickly by mappers, while the aggregation of histograms and construction of coarse region can be performed by a reducer. In the following MapReduce step, objects are mapped into their corresponding coarse regions, where each coarse region is handled by a reducer that will execute a more fine-grained partitioning on this region, with one of the six algorithms described above.

4. PERFORMANCE

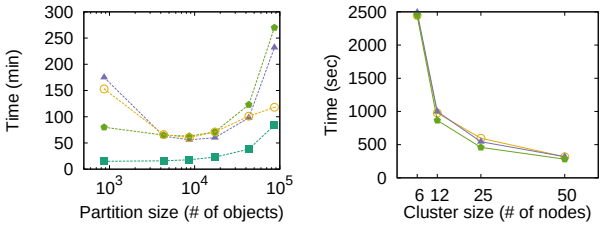
We use a 50 node Amazon Elastic MapReduce (EMR) cluster for our experiments. Each node is an extra-large EMR instance equipped with 15 GB memory, 4 virtual cores and 4 disks with 420 GB storage. S3 is used as the primary data storage system for data serving.

Datasets. We use two datasets for our experiments. The two datasets are from two different domains each with its own unique characteristics, which are representative of real world applications. The first dataset is *OpenStreetMap*[1]. It contains spatial representation of geometric features such as lakes, forests, buildings and roads. Spatial objects are represented by a specific type such as points, lines and polygons. The table schema is simple and it has roughly 70 columns. We use a subset of the OSM data which contains more than 87 million polygonal objects. The second dataset is an *Imaging* dataset from analytical pathology imaging studies for brain tumor research, where boundaries of micro-anatomic objects in the images are segmented and represented as polygons. We use a set of

[1]http://planet.openstreetmap.org

18 images, and each image contains 0.5 million spatial objects on average.

We use a spatial join query to evaluate the effect of partitioning on the query performance. Figure 3(a) shows the query processing runtime of join query with 4 different partitioning approaches and parameters. We can see from the figure that a proper partitioning approach can greatly increase query performance. Even for the same partitioning approach, the partitioning granularity can significantly affect the query performance.



(a) Join query performance (b) Parallel partitioning

Figure 3: Query processing time and parallel partitioning time

We also test the parallel partitioning performance on the OSM dataset with different cluster size configurations. Figure 3(b) shows the scalability test for parallel partitioning task. It is clear from the figure that, the parallel partitioning approaches have a very good scalability. Query tasks that require ad-hoc partitioning can benefit from such scalability, and overall query performance can be significantly improved.

5. DEMONSTRATION DETAILS

SATO runs on both local clsuters and Amazon Elastic MapReduce.

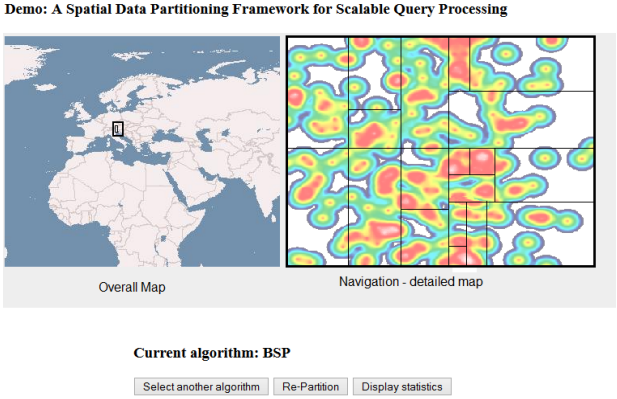


Figure 4: SATO web interface

The demo will show how the user navigates and uses SATO to effectively partition the input data. The system provides user interface with interactive results, and users can provide various parameters such as input dataset, sampling method, and sampling ratio. The user can optionally specify the sampling algorithm and associated parameters. The system executes multiple MapReduce jobs to sample and analyze the user data, and provides preliminary partition results in a format of heat maps with statistics. The user can scroll through dynamically generated visual maps and data summarization to select a final algorithm to generate the actual partitioned data. Data summarization includes object statistics of global, local partitions and their storage space requirement. Figure 4 shows a specific interface of the system used for data partitioning task.

The system can also provide a partition suggestion based on the dataset characteristics, and visually show various parameters to compare different approaches in color coded density map.

To demonstrate the difference between generated partitions and the use of multi-level indexing scheme, we also provide Hadoop-GIS as the spatial query processing system to utilize the output partitions from SATO. Hadoop-GIS will execute sample queries such as containment and spatial join queries on pipelined SATO outputs for different partitioning algorithms. The integrated system will compute and display the results, as well as measure computational cost. Here users can observe how query performance is affected by providing different result datasets generated by different partitioning approaches.

6. CONCLUSION

In conclusion, our SATO framework is a scalable generic solution that can effectively handle major issues, such as data-skew and irregularity of spatial objects, while optimizing future query performance. For comparison, SATO can produce high-quality partitioning on very large datasets such as OpenStreeMap polygons in less than 10 minutes with MapReduce support, while it might take hours for a standalone program to produce similar partitioning results. SATO is an open source system, which can run as standalone or be parallelized in MapReduce. Therefore, with the emergence of spatial Big Data, SATO is expected to attract interest from the open source community and its functionalities is to be further extended.

7. ACKNOWLEDGEMENTS

This work is supported in part by NSF IIS 1350885, by NSF ACI 1443054, by Grant Number R01LM009239 from the National Library of Medicine and by Grant Number 1U24CA180924-01A1 from the National Cancer Institute. We are also grateful for the support from Amazon AWS in Education Research Grant, and Google Summer of Code Project.

8. REFERENCES

[1] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *ACM SIGMOD Record*, volume 28, pages 13–24, 1999.

[2] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *Proc. VLDB Endow.*, 6(11):1009–1020, Aug. 2013.

[3] A. Eldawy. Spatialhadoop: towards flexible and scalable spatial processing using mapreduce. In *Proceedings of the 2014 SIGMOD PhD symposium*, pages 46–50. ACM, 2014.

[4] S. T. Leutenegger, M. A. Lopez, and J. Edgington. Str: A simple and efficient algorithm for r-tree packing. In *ICDE*, pages 497–506. IEEE, 1997.

[5] J. Lu and R. H. Guting. Parallel secondo: Practical and efficient mobility data processing in the cloud. In *Big Data*, pages 107–25. IEEE, 2013.

[6] O. O'Malley. Terabyte sort on apache hadoop. `http://sortbenchmark.org/Yahoo-Hadoop`, 2008.

[7] J. Patel et al. Building a scaleable geo-spatial dbms: technology, implementation, and evaluation. In *SIGMOD*, pages 336–347, 1997.

[8] S. Ray, B. Simion, A. D. Brown, and R. Johnson. A parallel spatial data analysis infrastructure for the cloud. In *SIGSPATIAL*, pages 274–283. ACM, 2013.

[9] B. Sowell, M. V. Salles, T. Cao, A. Demers, and J. Gehrke. An experimental analysis of iterated spatial joins in main memory. *Proc. VLDB Endow.*, 6(14):1882–1893, Sept. 2013.