



SGPAC: generalized scalable spatial GroupBy aggregations over complex polygons

Laila Abdelhafeez^{1,2} · Amr Magdy^{1,2} · Vassilis J. Tsotras^{1,2}

Received: 25 December 2020 / Revised: 3 November 2022 / Accepted: 2 February 2023 /
Published online: 21 March 2023
© The Author(s) 2023

Abstract

This paper studies the *spatial group-by query* over complex polygons. Given a set of spatial points and a set of polygons, the spatial group-by query returns the number of points that lie within the boundaries of each polygon. Groups are selected from a set of non-overlapping complex polygons, typically in the order of thousands, while the input is a large-scale dataset that contains hundreds of millions or even billions of spatial points. This problem is challenging because real polygons (like counties, cities, postal codes, voting regions, etc.) are described by very complex boundaries. We propose a highly-parallelized query processing framework to efficiently compute the spatial group-by query on highly skewed spatial data. We also propose an effective query optimizer that adaptively assigns the appropriate processing scheme based on the query polygons. Our experimental evaluation with real data and queries has shown significant superiority over all existing techniques.

Keywords Spatial big data · Group by · Query processing · Aggregation

1 Introduction

Spatial data is readily available in large quantities through the emergence of various technologies. Examples include user-generated data from hundreds of millions of users, fine-granularity satellite data from public and private sectors, ubiquitous IoT applications, and traffic management data. Such big spatial datasets are rich in information and come with new challenges for data scientists who try to explore and analyze them efficiently in various applications. These challenges span the whole stack of spatial data management, starting from revisiting fundamental queries and their variations to support the current volume scale efficiently.

In this paper, we address a spatial group-by query to efficiently support large-scale datasets that contain hundreds of millions or even billions of data points on real polygons

✉ Laila Abdelhafeez
labde005@ucr.edu

¹ Department of Computer Science and Engineering, University of California, Riverside, CA, USA

² Center for Geospatial Sciences, University of California, Riverside, CA, USA

with very complex perimeter geometries that have tens of thousands of points. Such overwhelming polygon complexity combined with large volume data poses challenges to the currently available techniques in the mainstream spatial data management systems. Given a set of spatial points and a set of complex polygons, our group-by query counts the number of spatial points within each polygon's boundaries. So, it groups points by polygon boundaries. This query is a composition of the fundamental spatial range query using sets of polygons as spatial group-by conditions. With real polygons, spatial containment checks are highly complex and consume significant processing overhead. For example, the polygons of provinces' borders worldwide have one million points on their perimeters [43]. Combining such complex boundaries with hundreds of millions of spatial data points encounters highly inefficient query latency that easily escalates to hours of processing on a dozen CPUs. These real polygons are heavily used by social scientists in various spatial statistical analysis applications, such as spatial regionalization [13], spatial harmonization [25], segregation analysis [9], join-count analysis, hot-spot, and cold-spot analysis, and spatial autocorrelation analysis [38].

The need for this research has been triggered during a collaboration with social scientists [3] to perform large-scale analysis on user-generated social media data at the world-scale. In that study, we analyzed *one billion tweets* through location quotients and spatial Markov analysis over complex spatial polygons. A fundamental operation for this analysis is aggregating counts of spatial points within each polygon. Due to the prohibitive cost of performing such aggregations, our study was limited to the US counties and a portion of the available Twitter datasets. To enable social scientists to perform large-scale studies at world-scale data, it is pivotal to efficiently support such fundamental operations on modern large-scale datasets for real complex polygons and multiple spatial scales.

Traditionally, count aggregations over polygons are performed using filter-refine approaches [17, 19]. The filter phase retrieves a subset of data based on the polygon minimum bounding rectangle (MBR). This subset is refined where each data point is tested against the exact polygon geometry using point-in-polygon checks. This approach is still used in modern distributed systems, e.g., Apache Sedona [46]. However, it incurs significantly expensive computations on large data. For example, running a single query for 100 million points over only 255 country borders takes an hour to finish, using a twelve-nodes Apache Sedona cluster with a total memory of 1TB. Such inefficient runtime limits spatial data scientists from performing large-scale analysis on modern spatial datasets.

Existing approaches face two challenges to handle modern large datasets efficiently. The first challenge arises from the prohibitive computations of point-in-polygon checks on real complex polygons due to the excessive number of points on the polygon perimeter. This is much higher than polygons arising in computer graphics applications [32, 41, 44], which have only tens of perimeter points. The second challenge is the high skewness of real spatial data due to skewed spatial distributions of the data generators, e.g., web users or city sensors. Such skewness leads to prohibitive runtime cost in parallel algorithms due to load imbalance.

To overcome these challenges, we propose the Spatial GroupBy Polygon Aggregate Counting (*SGPAC*), a highly-parallelized query processing framework to efficiently support spatial group-by queries in mainstream spatial data management systems. The *SGPAC* framework can efficiently aggregate counts for large-scale datasets over a large number

of highly complex polygons. To this end, *SGPAC* crumbles both data points and query polygons into fine-granular pieces based on two-level spatial indexing. For real polygons, a two-level *clipper* significantly downsizes the number of perimeter points to considerably speed up computing group-by aggregates while ensuring exact results. The counting process over the new clipped polygons is highly-parallelizable and makes great use of the distributed computation resources. Nevertheless, even with parallelism, *SGPAC* still encounters high computations on a few nodes –due to spatial data skewness– that can dominate the overall runtime. To address this, *SGPAC* identifies highly skewed data partitions to be further divided and directs data to underutilized nodes for distributed load balancing.

To support efficient performance on various query workloads, we propose a query optimization technique that distinguishes query polygons that are simple enough for which a plain filter-refine approach would suffice (i.e., *SGPAC* adds unneeded overhead). We thus present cost models for the *SGPAC* clipping approach and the filter-refine;

based on these models, we produce a cheap query plan that adaptively assigns an appropriate processing scheme for each query polygon.

Furthermore, our query processing and optimization techniques are generalized for any underlying distributed spatial index structures. This makes them plug-and-play modules in a wide variety of applications and systems without a need to change the underlying system infrastructure. We performed an extensive experimental evaluation with real spatial data and world-scale polygons. Our techniques have shown significant superiority over all existing techniques for real complex polygons.

This paper extends our work in [1]. The new extensions include (1) Handling highly-skewed spatial dataset through one-to-many spatial partitioning. (2) Building a cost-based query optimizer that selects the cheapest query plan for various query workloads. (3) Generalizing the query processing framework to work with various underlying indexes. (4) Enriching the experimental evaluation with eighteen synthetically-composed polygon datasets to study the effect of different parameters under controlled values. (5) Evaluating the effect of tuning the global index and the local index on the query performance, in addition to evaluating the new query optimizer. Our contributions are summarized as follows:

1. We define a challenging spatial group-by query (*SGPAC*) over real complex polygons.
2. We propose a novel distributed query processing framework to efficiently support the *SGPAC* query and extend it to support highly skewed datasets.
3. We introduce cost models that enable a query optimizer to produce the best query plan for processing the *SGPAC* query.
4. We provide generalizations for both the query processing framework and the query optimizer to work in numerous systems with various index structures.
5. We perform an extensive experimental evaluation using 100M real Twitter posts and up to 45K real spatial polygons organized in both real synthetically-composed layers with tuning both the local and the global indexes.

In the rest of this paper, we outline related work in Sections 2, while 3 formally defines the problem. The proposed query processing and query optimizer are detailed in Sections 4 and 5, respectively. Section 6 discusses our framework generalizations. Sections 7 and 8 present experimental evaluation and conclusions.

2 Related work

Centralized techniques Aggregation over spatial polygons has been studied for long and several techniques have been proposed [4–7, 16, 17, 19, 24, 26–31, 40, 48, 50]; the majority of them produce exact results while there are also works that produce approximate results, e.g., [5, 26, 47]. The most widely-used techniques are based on the two-phase filter-refine approach [17, 19] that filters out irrelevant data based on polygon approximations, most commonly a minimum bounding rectangle (MBR) approximation, and then refines candidate points based on polygon perimeter geometry. Several works proposed more precise polygon approximations in the filtering phase. Brinkhoff et al. [6] studied different approximations, namely rotated minimum bounding box (RMBB), minimum bounding circle (MBC), minimum bounding ellipse (MBE), convex hull (CH), and minimum bounding n-corner (n-C). Sidlauskas et al. [40] improved filtering by clipping away empty spaces in the MBR. Other approaches proposed multi-step filtering [7, 27, 28] and rasterization-based polygon approximation [4, 5, 16, 47, 50] to reduce the candidate set size. Hu et al. [21, 22] proposed indexing the line segments forming the polygon with an in-memory R-tree index, and then using this index to facilitate both the filter and the refinement steps. The points in the input dataset are indexed using another R-Tree index. The polygon R-tree index is used to filter out index nodes from the point R-tree that are outside the polygon. The index nodes from the point R-tree that are fully included within the polygon are provided in the result set along with all their children. The costly refinement operation is performed for index nodes from the point R-tree that intersect with the polygon. Although the rich approximations result in a tight candidate set, they do not reduce the computation complexity of each point-in-polygon check, which depends on the number of polygon perimeter points. Kipf et al. [26] solves an orthogonal point-polygon join query that takes a single point and outputs polygons that contain this point. This work assumes that all polygons are known beforehand, and thus can be indexed, while the data points are streamed. This is a majorly different setting than our setting that considers an arbitrary set of polygons as query input to query stationary data of large volumes.

Another direction in supporting polygon aggregation is polygon decomposition [29–31, 35]. Such decomposition reduces the complexity of a polygon query by dividing it into smaller polygons. The original polygon geometry is decomposed into regular forms, e.g., convex polygons, triangles, trapezoids, combinations of rectangles and triangles, or even smaller irregular polygons using a uniform grid. However, decomposition has not yet been adopted in distributed big spatial data systems. This is confirmed by recent surveys [15, 36] that examined work on range queries in modern big spatial systems. Current big spatial systems mostly focus on rectangular ranges and have limited support for arbitrary polygons with complex shapes and high-density perimeters, which is crucial for data analysis on real datasets, e.g., in social sciences, since real-world polygons are neither rectangular nor regular.

Distributed and parallel techniques There are also recent works on addressing irregular polygon range queries using distributed partitioning techniques [18, 33, 34, 37] and parallel GPU-based techniques [2, 47, 49]. These techniques mostly rely on partitioning data across a cluster of machines or GPU cores so that one query is partitioned along with data partitioning, and then the query executes on multiple nodes/cores that have relevant data. Nodarakis et al. [34] partition the data based on either a regular grid or angle-based partitioning. However, that work only supports convex polygons. Guo et al. [18] partition the

data using a quadtree, then different partitions work in parallel using a traditional filter-refine approach. Ray et al. [37] partition the data based on either object size or point density to improve workload balance among nodes. Malensek et al. [33] proposed bitmap-based filtering where the global view facilitates node selection based on the intersection with query polygons. While these distributed approaches tighten the candidate set and take advantage of parallelism, they do not reduce the computational complexity of point-in-polygon checks, which are the bottleneck. In addition, in many cases, polygons that lie within the boundaries of one partition do not make use of candidate set reduction at all. Therefore, even in distributed environments, their overall computational cost is still high on large-scale datasets and large real polygon sets. Also, GPU-based techniques are not widely incorporated in the mainstream spatial systems.

Our work follows the decomposition direction in distributed environments to speed up polygon aggregations. Compared to existing literature, our work is distinguished by (a) Inherently considering real complex polygons and large-scale datasets (that contain hundreds of millions of points) by identifying and reducing the main performance bottleneck. (b) Addressing high skewness in spatial data that leads to a high imbalance in distributed workloads. (c) Leveraging our novel techniques with existing techniques through a novel query optimizer to produce the best query processing plan.

3 Problem definition

Consider a spatial dataset D that consists of point objects. Each object $o \in D$ is represented by $(oid, lat, long)$, where oid is the object identifier and $\langle lat, long \rangle$ represent the latitude/longitude coordinates of the object's location in the two-dimensional space. Formally, an *SGPAC* query q is defined by a set of polygons L as follows:

Definition 1 (Spatial GroupBy Polygon Aggregate Counting (SGPAC) Query) Given a spatial dataset D , a query q defined by a set of polygons $L = \{l_1, l_2, \dots, l_m\}$, returns a set of m integers $\{c_1, c_2, \dots, c_m\}$, where c_i is the number of objects $o_j \in D$ so that o_j 's location lies inside polygon $l_i \in L$.

Each polygon $l_i \in L$ is represented by n_i spatial points that define its perimeter geometry. For large values of m and n_i and a large-scale spatial dataset D with hundreds of millions of points, scaling a spatial group-by query is highly challenging.

As with typical relational group-by queries, the query groups (in our case the polygons in L) can be selected in an ad-hoc manner. The polygon sets that are used in different applications, e.g., social sciences, could be either pre-defined polygons, e.g., US states borders, or arbitrary polygons, e.g., polygons produced by regionalization and harmonization algorithms [13, 25]. Here we assume the general case, where set L is not known apriori (and thus cannot be indexed beforehand). Since groups in a typical group-by clause are disjoint, the polygon set consists of disjoint polygons. However, our processing algorithm can also support polygon sets with overlapping polygons.

4 Query processing

We proceed with Section 4.1 which presents the generic query processing framework on which *SGPAC* is built, followed by Section 4.2 which discusses the details of the framework

instantiation based on a global quadtree index, and Section 4.3 that considers two variants for the local indexes, namely a grid index and an R-tree index.

4.1 Query processing framework

The rationale behind this framework depends on two observations. The first observation recognizes that the main performance bottleneck of polygon aggregations is the high computational cost of point-in-polygon checks. So, our framework relies on significantly reducing the complexity of these checks to minimize the overall cost of large polygon sets aggregations. The second observation is that in nowadays applications, large-scale spatial datasets are usually indexed on distributed big spatial data systems, such as Apache Sedona [46], Simba [45], GeoMesa [23], or SpatialHadoop [14]. Therefore, we develop a query processing framework that exploits such distributed indexing infrastructure in reducing the computational cost of spatial group-by queries to make our techniques applicable to a wide variety of existing applications and platforms.

Figure 1 shows an overview of our query processing framework. The framework exploits partitioning, by facilitating a global distributed spatial index to partition both data points and query polygons across different machines (distributed worker nodes) using standard methods as in [14]. Each worker node j covers a specific spatial area represented with a minimum bounding rectangle (MBR) B_j . Then, on each worker node, the local portion of data points is indexed with a local spatial index, which does not necessarily have the same structure as the global index. The local index further divides data into small chunks. Meanwhile, when a new query polygon set L arrives, each worker node receives a subset L_j of query polygons that overlap with its partition MBR B_j ; that is, for all $l_i \in L_j, l_i \cap B_j \neq \phi$. Each polygon $l_i \in L_j$ goes through a *Two-level Clipper* module that significantly reduces the complexity of its perimeter through two phases of polygon clipping. The first phase is based on the global index partition boundaries B_j . This phase replaces l_i with $l_i \cap B_j$, its intersection with the partition MBR, as any part of the polygon outside B_j will not produce any results from the data points assigned to node j . The newly clipped polygon l_i is passed as an input to the second level of clipping, which further clips l_i based on the local index partitions to produce multiple smaller polygons, each of them corresponding to one of the local index partitions and clipped with its MBR boundaries.

After the two-level clipping of input polygons, the query input turns into small crumbles of both local data partitions and simple query polygons that are fed to a multi-threaded *Point-in-Polygon Refiner* module. This module takes pairs of data partitions and clipped query polygons with overlapping boundaries, where each pair follows one of two cases. The first case is that the boundaries of both the local partition and the clipped query polygon are the same. This means that the local partition is wholly contained inside the query polygon and all data points are counted in the result set without further refinement. The second case is that the clipped query polygon intersects with part of the local partition boundaries. In this case, the refinement module iterates over all the points within the local partition and simply uses the standard point-in-polygon algorithms (as in [20]) to filter out points that are outside the polygon boundaries. Such point-in-polygon operation is much less expensive on the clipped polygon compared to the original polygon, with up to an order of magnitude cost reduction as shown in our experiments. Each thread maintains a list of $\langle \text{polygonid}, \text{count} \rangle$ pairs that record the count of points in each polygon. Lists of pairs from different threads and partitions are forwarded to a shuffling phase that aggregates total counts of each input polygon, based on polygon ids, in a similar fashion to the standard map-reduce word counting procedure.

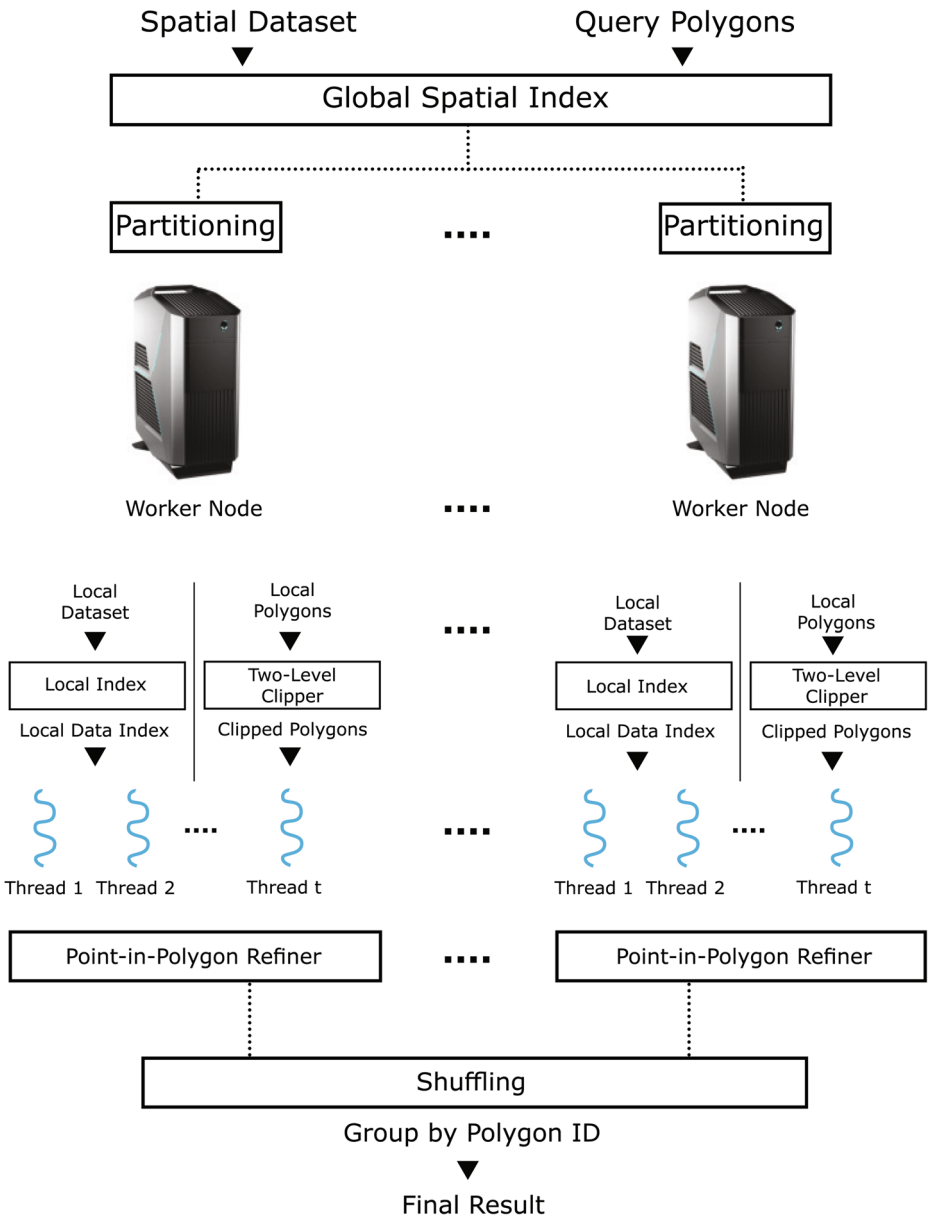


Fig. 1 Query processing framework overview

This query processing framework puts no assumptions on the underlying index structures, both global and local indexes, so it can be realized in a wide variety of existing applications and platforms that already use distributed spatial indexing in their data management pipelines. However, the skewness of spatial data may introduce further challenges to this query processing framework. In Sections 4.2 and 4.3 we discuss instantiating the

SGPAC query processor based on this framework and adjusting the framework modules for highly-skewed spatial datasets.

4.2 *SGPAC* global index instantiation

We present an instantiation of the *SGPAC* query processor where in order to handle highly skewed data, the global index is a quadtree structure, because of its ability to adapt for high skewness in data (common in several spatial datasets) through adapting the tree depth in different spatial areas based on data density. Given a parameter *capacity* that defines how many points are allowed within a quadtree partition, the quadtree partitioner starts by inserting the whole dataset in the root tree node. If the node capacity is exceeded, it is divided into four child nodes with equal spatial area, and its data is distributed among the four child nodes. If any of the child nodes has exceeded its capacity, it is further divided into four nodes recursively and so on, until each node holds at most its parameterized *capacity*. With this standard mechanism, spatial areas with high densities of data are further divided into deeper tree levels, while sparse areas will result in shallow tree depth. The optimal goal is to hold an equal data load in each partition, which leads to balancing the distributed query processing time when this data is processed for incoming queries.

Nevertheless, for modern real datasets, a straightforward quadtree adaptation is not enough to handle data skewness. One problematic case that is common in user-generated spatial datasets is having dense hotspots (i.e., *high-density points*) rather than high-density areas. A hotspot corresponds to a single point in space that has a high concentration of data. Data hotspots have become more common recently due to privacy concerns that encourage platforms to geotag spatial data with broad locations, e.g., cities, rather than precise points. Similarly, most users prefer not to disclose their GPS location and instead mark their tweets with the user's home location (for example 'Los Angeles', thus assigning to a tweet the GPS of the city's center). The traditional quadtree cannot handle such cases since an overcapacity node with a high-density point will continue to split infinitely or stop at a maximum level while still holding an excessive number of points. Such overloaded nodes cause considerable load imbalance, which significantly skews runtime leading to highly inefficient query processing time.

To overcome such a severe limitation, our *SGPAC* query processor modifies the standard quadtree partitioner to detect high-density points and enable a one-to-many partitioning scheme. The one-to-many partitioning scheme allows the data points of an overloaded (skewed) partition to be divided among many partitions that share the same boundaries; such partitions can be sent to different worker nodes and thus improve the workload balance. In specific, to detect high-density points, while splitting a quadtree node, the new partitioner checks if data skewness spans multiple successive levels. An integer counter *skewed_levels* is initialized to zero for each tree node. On splitting a node, when all data points go to only one child node, its *skewed_levels* counter is incremented by one. Once *skewed_levels* reaches a threshold, the split stops at this level and the partition is marked as a skewed partition. Each skewed partition is then divided into multiple partitions, each has the same spatial boundaries as the original partition but containing a distinct portion of the dataset (of size equal to the parameterized node capacity). These multiple partitions are stored with different partition ids and machine ids in the global index metadata, so they are seamlessly considered by the query processor during the polygon partitioning and shuffling phases.

Later in Section 6, we discuss other potential instantiations based on different spatial index structures to serve other potential spatial datasets.

4.3 SGPAC local index instantiation

In this Section, we present two different local indexes instantiations, the first is based on a grid index, and the other is based on an R-tree index.

(I) Grid Local Index

At the local indexing level, the first available instantiation in SGPAC is using a regular spatial grid index that divides the space into equal-sized spatial tiles. The grid index provides an efficient yet simple way to index the data locally and still serves the purpose of crumbling query polygons into small pieces. Each grid cell in the local grid index serves as a local partition, which is passed along with the clipped simpler polygons to the multi-threaded *Point-in-Polygon Refiner* module.

(II) RTree Local Index

Another possible instantiation of the local index is the usage of R-tree index that organizes the data in a balanced tree according to their spatial proximity, and data are stored at the leaf nodes of the tree. Each non-leaf tree boundary is used to clip the original query polygons. The clipping starts at the root node, and each parent node passes to its children the part of the polygon that it intersects with. Each leaf node serves as a local partition, which is passed along with the clipped simpler polygons to the multi-threaded *Point-in-Polygon Refiner* module. The R-tree works better for SGPAC for two main reasons: (1) Irrelevant nodes can be pruned earlier at a higher-tree level, saving filtering time. Instead of checking all irrelevant leaf nodes, checking their parents is sufficient to prune them. (2) The cost of clipping the polygon is reduced at each level of the tree since a parent node passes a clipped version of the query polygon to its children.

5 Query optimization

Our group-by query will be supported as a system operator having an impact on a wide user base, e.g., in distributed big spatial data systems such as Apache Sedona or parallel spatial database systems, such as PostGIS. From a system builder perspective, the system expects various query workloads and identifies a cheap query plan for efficient execution, which is the main objective behind the long history of query optimization work. Our SGPAC technique is optimized for processing complex polygons, which is the case for most real spatial datasets. However, in the case of simple polygons, a system should be able to decide on using a simpler technique, e.g., a traditional filter-refine approach, to execute such queries efficiently.

To help the optimizer decide whether to use the traditional filter-refine approach or to pay the overhead of the SGPAC technique, we develop theoretical cost models that calculate an approximate computation cost for each query polygon in a distributed fashion based on polygons that are partitioned across worker machines. The estimated cost in each partition depends on its local polygons and local data distribution. Both cost models depend on estimating the cost of the steps after the initial filter step that excludes data from irrelevant distributed partitions. This step is common for both techniques and is implicitly included in the global index partitioning, so it not included in the cost comparison.

5.1 Filter-refine cost model

A filter-refine approach refines all data points that lie within the polygon MBR. If n denotes the number of perimeter points in the polygon, the time complexity of a single point-in-polygon check is $O(n)$. Therefore, the overall refinement cost for a polygon is $C_{fr} = d \times n$, where d is the number of data points in the polygon's MBR. If the data is uniformly distributed, then $d = \frac{A_{MBR}}{A} \times |D|$, where A_{MBR} is the area of the polygon MBR, A is the total area of the covered spatial space, and $|D|$ is the cardinality of the underlying dataset D . However, real spatial data is highly skewed and we can not make this uniformity assumption. Moreover, the polygon might span multiple worker machines, which in turn reduces the overall cost since the refinement runtime of the whole polygon is distributed among multiple data partitions. To handle data skew and multi-machine split, we introduce a distributed count estimator data structure associated with each local index. This estimator data structure is of a spatial grid data structure, but instead of storing actual data points, it stores the number of data points that lie within the boundaries of each grid cell. If we index the data locally using a local grid index, generating the count estimator data structure is straightforward. In this case, the count estimator grid cell granularity follows the same granularity of the local data index. However, having the data locally indexed using any tree-based spatial index, introduces some challenges in generating the count estimator data structure. The first straightforward option is to have the count estimator data structure follow the structure of the local index. However, the grid data structure is simpler and is easier to maintain and access. For this reason, we advocate for keeping the estimator data structure as a grid regardless of the underlying data index. In cases of structure mismatch, we map the data count information from the original data structure to the grid data structure of an arbitrary cell size r . The details of this mapping are further discussed in detail in Section 6.2.

After generating the estimator data structure, when computing the cost within each partition, we access this estimator to estimate the number of data points that lie within polygon MBR. For a set of distributed data partitions P , the cost within each local partition $P_k \in P$ is estimated by the following formula:

$$C_{fr}^k = \left(\sum_{j=Y_{min}}^{Y_{max}} \sum_{i=X_{min}}^{X_{max}} c_{ij} \right) * n \quad (1)$$

Where c_{ij} is the count within the estimator grid cell (i, j) , and $Y_{min}, Y_{max}, X_{min}, X_{max}$ are indices that give the four corners of the polygon MBR mapped on the grid. This formula sums over all cells that overlap with the polygon MBR within the local partition P_k . Although the overall cost of a single polygon query is the maximum among all partitions, $C_{fr} = \text{Max}(C_{fr}^k), \forall P_k \in P$, the optimization decision is actually taken locally regardless of other partitions. This cost model for the filter-refine operation defined in (1) is the same for all local index variants.

5.2 SGPAC cost model

Contrary to the filter-refine cost model mentioned in Section 5.1, the SGPAC cost model is affected by the nature of the underlying index. In the following Sections 5.2.1 and 5.2.2, we discuss the cost model for the two variants of the local index instantiations mentioned earlier in Section 4.3. Similar to the filter-refine model, equations mentioned in the following Sections (2) and (3) estimate the cost for a single polygon on a single partition $P_k \in P$. The overall estimated cost for this polygon query in a distributed environment is given by,

$C_{SGPAC} = \text{Max}(C_{SGPAC}^k), \forall P_k \in P$. Again, the global cost C_{SGPAC} is not actually used as each local partition decides on its optimal processing strategy locally.

5.2.1 Grid local index cost model

The *SGPAC* approach performs multiple refinement operations in each local partition at multiple grid cells of the local grid index. Each refinement operation uses a smaller polygon fragment, compared to the original polygon, after the two-level polygon clipping. To model the cost of refinements, we assume: (a) the whole original polygon perimeter lies within the local partition boundaries, (b) the number of local index grid cells that overlap with the polygon MBR is g_c , (c) each of the overlapping g_c grid cell has a polygon fragment of $\frac{n}{g_c}$ perimeter points, where n is the number of the polygon perimeter points, and (d) a refinement operation is performed for each of the g_c cells, which is a conservative assumption as in practice some of these cells are wholly contained within the polygon and do not perform refinement as detailed in Section 4.

Given the above assumptions, the cost of *SGPAC* query processing on a local partition P_k , C_{SGPAC}^k , is composed of two components: (i) polygon clipping cost C_{clip} , and (ii) point refinement cost C_{refine} . Therefore, $C_{SGPAC}^k = C_{clip} + C_{refine}$. The cost of a single clipping operation is $O(n \times \log(n))$ [39]. This operation is repeated g_c times, so the total clipping cost C_{clip} is estimated as $C_{clip} = g_c \times n \times \log(n)$. For the refinement cost, we use the same count estimation data structure that is used in the filter-refine cost model to estimate the number of data points within a polygon. This makes the refinement cost $(\sum_{j=Y_{min}}^{Y_{max}} \sum_{i=X_{min}}^{X_{max}} c_{ij}) \times \frac{n}{g_c}$, using the same notations as in (1). This is performed through a multi-threaded refinement module that uses t threads in parallel. So, the overall refinement cost is estimated by $C_{refine} = \frac{(\sum_{j=Y_{min}}^{Y_{max}} \sum_{i=X_{min}}^{X_{max}} c_{ij}) \times \frac{n}{g_c}}{t}$. Consequently, the overall estimated cost C_{SGPAC}^k is given by:

$$C_{SGPAC}^k = g_c \times n \times \log(n) + \frac{(\sum_{j=Y_{min}}^{Y_{max}} \sum_{i=X_{min}}^{X_{max}} c_{ij}) \times \frac{n}{g_c}}{t} \tag{2}$$

5.2.2 R-Tree local index cost model

The *SGPAC* approach starts with the two-level clipping of the query polygons. Each non-leaf tree boundary is used to clip the original query polygons. The clipping starts at the root node, and each parent node passes to its children the part of the polygon that it intersects with. Then *SGPAC* performs the refinement operations for each leaf node. *SGPAC* with a tree-based local index differs from *SGPAC* with a grid index in: (i) hierarchical clipping operations instead of a flat clipping with all intersecting nodes. This means that the clipping operation is not necessarily performed on all the tree nodes. (ii) clipping operation cost is significantly reduced at each level of the tree since a parent node only passes a clipped version of the query polygon to its children.

To model the total cost of *SGPAC* we reuse the first two assumptions mentioned for the grid index structure: (a) the whole original polygon perimeter lies within the local partition boundaries, (b) the number of local index grid cells that overlap with the polygon MBR is g_c , and we add the following assumptions: (c) at each level of the tree, the perimeter of the polygon is reduced by the branching factor of the tree b , so the polygon is of $\frac{n}{b^l}$

perimeter points at each level l , at $l = 0$ the root node, the polygon has n perimeter points, (d) since the cost of the clipping operation is $O(n \times \log(n))$, and since the polygon has a different value for n at every level l of the tree, we assume the average cost of the clipping operation to be the cost of clipping the polygon at the mid-height of the tree $m = H/2$ where H is the maximum height of the tree, making the average cost of one clipping operation $O(\frac{n}{b^m} \times \log(\frac{n}{b^m}))$, (e) since the refinement operation is only performed at the leaf nodes, if the polygon is evenly distributed amongst all leaves, the cost of the refinement operation would be $O(\frac{n}{b^H})$ and if the polygon only intersects with one leaf node the cost will remain $O(n)$, on average we assume the polygon is distributed amongst b^m nodes, where m is the mid-height of the tree.

Given the above assumptions, the cost of *SGPAC* query processing on a local partition P_k , C_{SGPAC}^k , is composed of two components: (i) polygon clipping cost C_{clip} , and (ii) point refinement cost C_{refine} . Therefore, $C_{SGPAC}^k = C_{clip} + C_{refine}$. To account for the hierarchical clipping we assume that the clipping operation cost is the average cost of clipping $O(\frac{n}{b^m} \times \log(\frac{n}{b^m}))$. This operation is repeated g_c times, so the total clipping cost C_{clip} is estimated as $C_{clip} = g_c \times \frac{n}{b^m} \times \log(\frac{n}{b^m})$. For the refinement cost, we use the same count estimator data structure that is used in the filter-refine cost model to estimate the number of data points within a polygon. This makes the refinement cost $(\sum_{j=Y_{min}}^{Y_{max}} \sum_{i=X_{min}}^{X_{max}} c_{ij}) \times \frac{n}{b^m}$. Consequently, the overall estimated cost C_{SGPAC}^k is given by:

$$C_{SGPAC}^k = g_c \times \frac{n}{b^m} \times \log\left(\frac{n}{b^m}\right) + \left(\sum_{j=Y_{min}}^{Y_{max}} \sum_{i=X_{min}}^{X_{max}} c_{ij}\right) \times \frac{n}{b^m} \tag{3}$$

6 Framework generalization

This section discusses the generality of our proposed framework, for both query processing and optimization, to work with various spatial applications even if they employ different spatial index structures and have different data characteristics. Section 6.1 highlights the generality of query processing, and Section 6.2 discusses the generality of query optimization.

6.1 Variant query processor instantiations

The presented *SGPAC* instantiations in Sections 4.2 and 4.3 are some of the possible instantiations based on our query processing framework (Section 4.1). However, the possible instantiations are not limited to the presented ones. In fact, the main advantage of our query processing framework is the ability to work with any underlying spatial indexing. For example, if an application already employs a spatial index such as R-tree or kd-tree, either on the global or local level, the same framework can be used without any significant changes. Only two differences are implied by having different indexes. First, the two-level polygon clipping module (in Fig. 1) will use the new index partitions. Second, the skewness detection technique (detailed in Section 4.2) will depend on the split/merge criteria of the underlying index. Other than that, every step in our query processing framework is intact and can adapt to various spatial indexes. Such a plug-and-play module enables spatial group-by queries to be supported efficiently in various applications and platforms.

6.2 Variant cost models

The other module that is affected by changing the underlying spatial index is the proposed cost models in Section 5. These models are mainly designed to handle skewed spatial data, so they depend on local spatial grid structures that are used as count estimators for data distributions within each local worker machine. Each grid estimator has the same granularity as the local spatial grid index, so it is straightforward to aggregate counts in each estimator cell from the corresponding index cell. However, if the underlying local index changes to a spatial tree index, e.g., quadtree, R-tree, or kd-tree, this estimator should respond to this change. One way to respond is to change the estimator structure from a grid to a tree with the same local index structure and directly aggregate counts from index cells. However, we do not advocate for such design choice and see it as overhead from a system perspective as estimators should be light to maintain and update, which is not the case for most tree structures compared to a simple grid estimator. So, we argue that system builders should keep the grid estimators even if their local indexes are not grid indexes. To overcome the problem of structure mismatch, we propose a generic way to generalize building our grid estimators for any spatial index structure.

The main challenge for the grid estimator is choosing an effective and efficient grid cell size, so the grid can be efficiently maintained and still provides accurate estimates. The granularity of the estimator cell size is left as a parameter for system administrators, enabling human-friendly system performance tuning. Given that, the value of r is a trade-off between the system overhead for maintaining estimators and the estimation accuracy. If r value is small, then the grid estimator cell size is small and data is aggregated in a large number of cells that provide high estimation accuracy with high maintenance overhead. If r value is large, then the grid size cell is large and data is aggregated in a low number of cells with low overhead, yet it provides lower accuracy as well. System administrators can thus tune r value based on the available system resources and desired accuracy. In Section 7.5 we show accuracy results while varying the granularity of the grid cell estimator.

Moreover, in our cost models, estimating the cost does not only depend on counting data within polygons but also depends on how the polygon intersects with local index cells. In our *SGPAC* first instantiation, this is straightforward because the estimator cells have one-to-one mapping to index cells. So, while counting data one can easily calculate the grid-polygon intersection; specifically, the value of g_c in (2) and (3) that represents the number of local index partitions that intersect with the query polygon. If the local index is not a grid index, but a spatial tree structure, then there is no one-to-one mapping between estimator grid cells and index cells anymore. In this case, to calculate g_c , the estimator grid cells do not only store data counts, but also store ids of corresponding index cells. Each estimator grid cell stores a list of ids for index cells that overlap with its boundaries. Then, g_c can be calculated as the total number of distinct index cell ids that overlap with the polygon MBR.

7 Experimental evaluation

This section presents an experimental evaluation of our techniques. Section 7.1 presents the experimental setup. Then, Sections 7.2–7.4 evaluate query processing scalability, index tuning, and query optimization effectiveness. Section 7.5 contains experiments on cost model evaluation.

7.1 Experimental setup

We evaluate the performance of *SGPAC* in terms of query latency using a real implementation based on Apache Sedona [46]. Our parameters include polygon count in the query polygon set (i.e., set cardinality), the average number of points per polygon perimeter (representing the complexity of polygon geometry), average polygon area, global quadtree partition capacity, local grid granularity and local R-tree capacity. Unless mentioned otherwise, the default value for the global quadtree capacity is 30K points, the default granularity is 10KMx10KM for the grid local index, and the default node maximum capacity is 1K points for the local R-tree index. Other parameters are experimentally evaluated and changed as illustrated in each experiment. All experiments are based on Java 8 implementation and using a Spark cluster of a dual-master node and 12 worker nodes. All nodes run Linux CentOS 8.2 (64bit). Each master node is equipped with 128GB RAM, and each worker node is equipped with 64GB RAM. The total number of worker executors on the Apache Spark cluster is 84, each with 4 GB of memory, along with an additional executor for the driver program.

Evaluation datasets Our evaluation data is a Twitter dataset that contains 100 million real geotagged tweets spatially distributed worldwide. For query polygons, we use real polygons from Natural Earth (NE) [11, 12] (collected through the UCR STAR data repository [42]), GADM [10] and ArcGIS [8]. The real polygon sets represent four different multi-scale spatial layers representing borders of continents, countries, provinces, and counties worldwide. Details of each set are shown in Table 1, where n is the number of perimeter points. However, these real layers have high variance in polygons' characteristics and counts. For example, in the counties, the number of perimeter points n ranges from very simple polygons with only four points to extremely complex ones, e.g., Baffin, Nunavut in Canada has 315K points. Due to this high variance in characteristics, we synthetically composed 18 polygon sets out of the same real polygons that compose the real layers. The synthetically-composed polygon sets have much less variance, almost homogeneous, in polygon count, areas, or perimeter complexity. These sets are used to study the effect of each parameter isolating the effects of other parameters. Table 2 shows characteristics of the 18 synthetically-composed polygon sets, where n is the number of polygon perimeter points. All sets are subsets from the real polygon layers that represent continents, countries, provinces, counties worldwide. These subsets are selected to vary one parameter and fix the other two so our experiment isolates the effects of each parameter. As shown in Table 2, the sets are divided into three groups. The first group contains six sets, L1-L6, that have increasing perimeter complexity, but the same polygon count and almost homogeneous areas per set. The second group contains six sets, L7-L12, that have increasing polygon count, but homogeneous perimeter complexity and areas per set. The third group contains

Table 1 Real Polygon Layers

	Count	Source	Min n	Max n	Average n	Average Area
Continents	6	ArcGIS [8]	2,904	16,206	9,454	42,493,878 km^2
Countries	255	NE [11]	4	22,907	1,345	1,122,947 km^2
Provinces	4,489	NE [12]	4	15,953	246	66,732 km^2
Counties	45,961	GADM [10]	4	315,455	925	5,710 km^2

Table 2 Synthetically-composed polygon sets

	Polygon Count	Minimum n	Maximum n	Average n	Average Area
L1	1,000	11	107	60	2,338 km^2
L2	1,000	107	488	250	7,102 km^2
L3	1,000	488	950	700	13,207 km^2
L4	1,000	2,000	3,500	2,500	37,367 km^2
L5	1,000	5,000	9,500	7,000	113,700 km^2
L6	1,000	9,500	315,455	18,000	609,506 km^2
L7	100	1280	1529	1,400	4,194 km^2
L8	500	1280	1535	1,400	46,856 km^2
L9	2,000	1280	1927	1,500	34,495 km^2
L10	4,000	1280	3415	2,000	35,931 km^2
L11	5,000	1280	5045	2,500	49,616 km^2
L12	10,000	731	9430	2,300	43,395 km^2
L13	1,000	732	8745	1,400	100 km^2
L14	1,000	731	8679	1,650	250 km^2
L15	1,000	732	9390	2,400	700 km^2
L16	1,000	732	9384	2,700	3,000 km^2
L17	1,000	732	9411	2,500	13,000 km^2
L18	1,000	732	9364	2,300	400,000 km^2

six sets, L13-L18, that have an increasing area, but the same polygon count and homogeneous perimeter complexity. The table shows the actual values of polygon count, perimeter points, and area.

Evaluated alternatives We evaluate our *SGPAC* technique (denoted as *SGPAC-2L*) against five alternatives: (1) A variation of *SGPAC* (denoted as *SGPAC-1L*) that employs only one-level clipping based on global index partitions and ignores local index clipping. (2) A variation of *SGPAC* (denoted as *SGPAC-QO*) that employs our query optimizer. (3) A distributed filter-refine approach (denoted as *FR*) that uses the popular MBR-based filter-refine (as discussed in Section 2) on each worker node in parallel. (4) A spatial join based approach (denoted as *SP-Join*) that partitions both data points and polygons based on the global index, and then performs nested loop spatial join on each worker node in parallel. (5) A distributed variant of Hu et al. [21, 22] approach that suggests indexing the query polygons in an in-memory R-tree index structure (denoted as *R-IDX*). All approaches are summarized in Table 3.

7.2 Query evaluation

Figures 2, 3, and 4 evaluate the query performance of the different techniques on the four real polygon sets of continents, countries, provinces, and counties (Table 1). The figures present the query performance for the two different local indexes, in ascending order of three different parameters: average number of perimeter points n per polygon (in Fig. 2), average polygon area per polygon (Fig. 3), and polygon count (Fig. 4).

Table 3 Evaluated Alternatives

Acronym	Approach
SGPAC-2L	SGPAC with two-Level clipping using local and global indexes
SGPAC-1L	SGPAC with only one-Level clipping using global index
SGPAC-QO	SGPAC with query optimizer
FR	Filter-refine approach
SP-Join	Spatial join approach
R-IDX	Indexing query polygons approach

In all cases, *SGPAC-QO* and *SGPAC-2L* outperform all other techniques. *SGPAC-1L* has significant performance enhancement compared to *FR* and *SP-Join* and comparable performance to *R-IDX*. The traditional *FR* or *SP-Join* only yields comparable performance in the provinces set which has the simplest polygon perimeters, with an average of 246 points per perimeter. Our proposed approach *SGPAC-2L* while using a local R-tree index takes under 1 minute to process all query polygons on the 100 million data points, except in one case where it takes under 3 minutes. When using a grid local index, *SGPAC-2L* takes 4 minutes to process these polygon sets.

In particular, Figs. 2, 3 and 4 gives the following major insights:

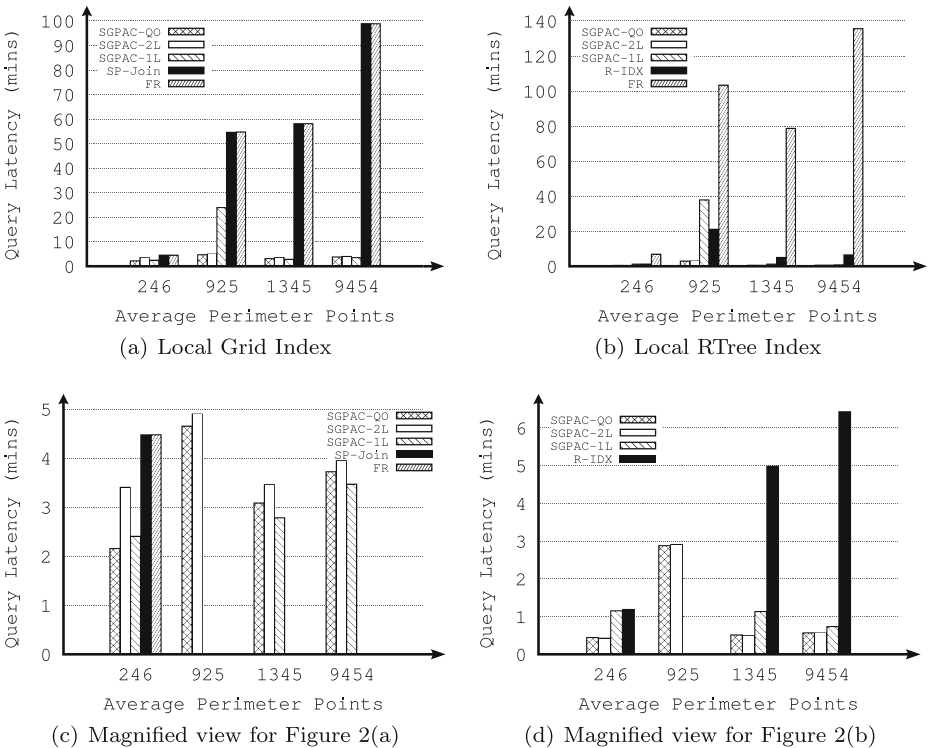


Fig. 2 Query performance on real polygon layers varying average perimeter points

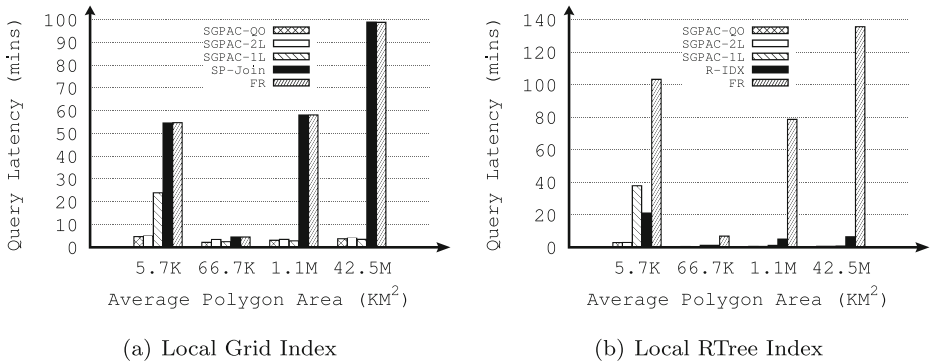


Fig. 3 Query performance on real polygon layers varying average polygon area

- (1) The first insight is the high variability in query speedup for different polygon layers. Specifically, as the average number of boundary points increases, the speedup significantly increases. Which means that for simpler polygon set, we see slight enhancement when using *SGPAC* variations, which is depicted in provinces set that has the simplest polygon perimeters, with an average of 246 points per perimeter; here *FR* and *SP-Join* perform comparably to *SGPAC* variations with only 2 times slower latency, this rate increases to be 15 times slower if *SGPAC* is incorporated with a local R-tree index. However, when the polygon complexity increases, with a higher number of perimeter points, *SGPAC* variations perform much better (up to 240 times faster) due to our proposed polygon complexity reduction. The effect of this reduction is significant and reduces query latency up to an order of magnitude on real complex polygons.
- (2) The second insight deals with the effectiveness of our optimization. In all cases, *SGPAC-QO* gives a similar or better performance compared to *SGPAC-2L*, which showcases the effectiveness of our query optimizer. In particular, as can be seen in

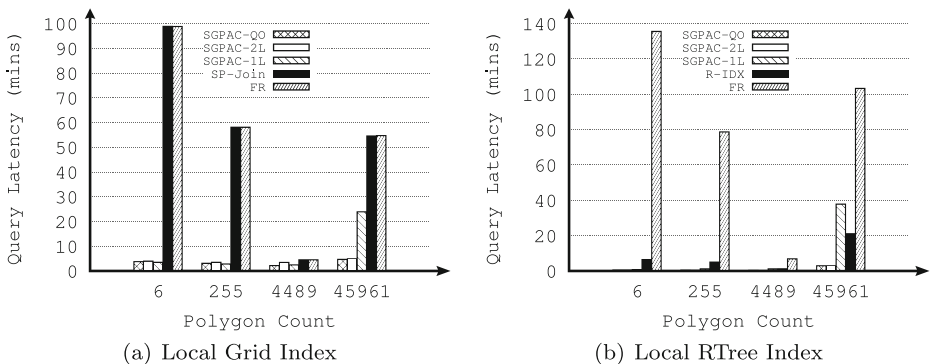


Fig. 4 Query performance on real polygon layers varying polygon count

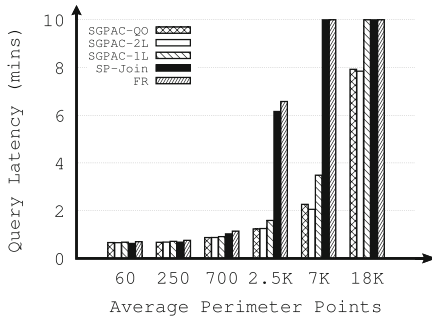
Fig. 2(c), *SGPAC-QO* shows its best performance over *SGPAC-2L* when polygons are the simplest (i.e. with an average of 246 points per perimeter). This means that *SGPAC-QO* cleverly employed a cheaper technique (filter-refine) when compared to *SGPAC-2L*. This showcases the effectiveness of our query optimizer in adaptively selecting the appropriate technique for various query workloads. Moreover, it is noticeable that the *SGPAC-QO* latency reduction is slight when compared to *SGPAC-2L* in all other cases. As real polygons are typically highly complex, in most cases the optimizer will use the two-level clipping approach (like *SGPAC-2L*) and rarely uses the filter-refine approach. This again confirms the superiority of our proposed two-level clipping technique in most real cases.

- (3) The third insight is the effectiveness of the second level of *SGPAC* two-level clipping for reducing perimeter complexity. As shown in Fig. 3(a) and (b), the *SGPAC-1L* approach (that ignores second level clipping) performs comparably to *SGPAC-2L* except in the case of counties that have the smallest area (5.7km^2), where *SGPAC-1L* performs five times slower than *SGPAC-2L* in case of using a local grid index, and 13 times slower if a local R-tree index is used. In that case, many county polygons are small enough to fit in the local machine without being clipped by the global index; that is, the polygon complexity is not reduced by the first level clipping. In such scenarios, ignoring the second level of clipping leads to much more expensive point-in-polygon checks and much higher overall latency.
- (4) When comparing Figs. 3 and 4 to 2, we do not see any clear effect in query latency from the increase in polygon area or polygon count. One possible interpretation is that area and polygon count might indeed have no effect, or that their effects are correlated. To further isolate the effect of these parameters, our following experiment uses the synthetically-composed polygon sets that contain polygons with controlled parameter values.

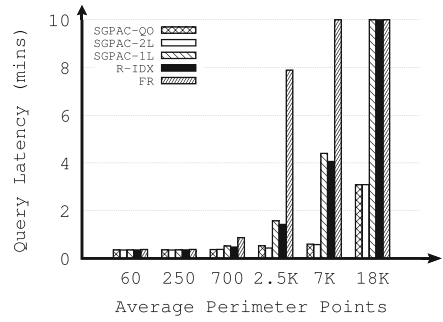
Figure 5 shows the same experiments of Figs. 2, 3, and 4 but on the synthetically-composed polygon sets that are described in Section 7.1. All sub-figures of Fig. 5 are truncated at 10 minutes. *FR* and *SP-Join* in Fig. 5(a) takes 21 minutes at 7K points, and 166 minutes at 18K, whereas *SGPAC-1L* takes 22 minutes at 18K. In Fig. 5(b), *FR* takes 27 minutes and 224 minutes for 7K and 18K points respectively. *SGPAC-1L* takes 32 minutes at 18K, whereas *R-IDX* takes 23 minutes. In Fig. 5(c), *FR* and *SP-Join* take 32–35 minutes at 400K KM^2 . Whereas in Fig. 5(d) *FR* takes 44 minutes at the same area. In Fig. 5(e), *FR* and *SP-Join* take 24 minutes at 5K points and 45 minutes at 10K points, whereas *FR* takes 32 minutes and 60 minutes at the same points in Fig. 5(f).

The results confirm that the *SGPAC* variations have significant performance superiority over other techniques and scale much better for large-scale datasets. Figure 5(a) gives a new insight that in simple polygons, *SP-Join* beats all other techniques, even *SGPAC-QO*. The reason is that our query optimization compares only *FR* and *SGPAC*, but does not consider *SP-Join*, so it never employs *SP-Join* even for very simple polygons.

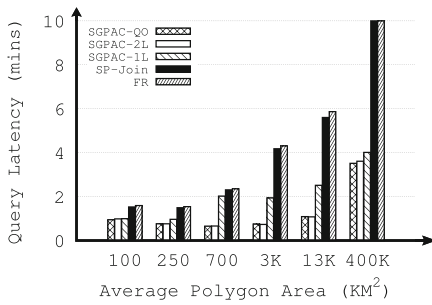
With respect to the polygon area parameter, Fig. 5(c) and (d), if one polygon set is much larger in area than another, it is likely that larger areas have higher query latency, due to a larger number of points inside polygons which requires more point-in-polygon checks. For the polygon count parameter, Fig. 5(e) and (f) show that when the number of polygons increases, query latency linearly increases for all techniques due to more point-in-polygon operations. That is, polygon count has a clear positive correlation with query latency. Yet, the *SGPAC* variations have the least increase in latency, while other techniques have a much higher increase.



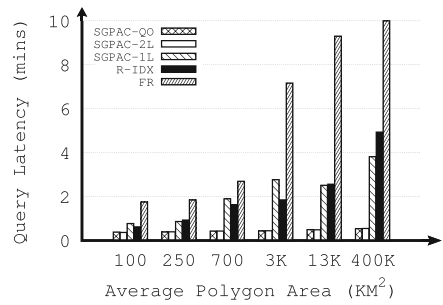
(a) Varying average perimeter points (Grid Index)



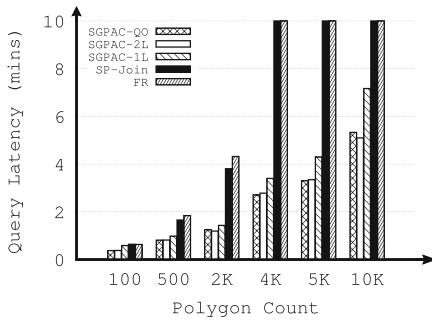
(b) Varying average perimeter points (RTree Index)



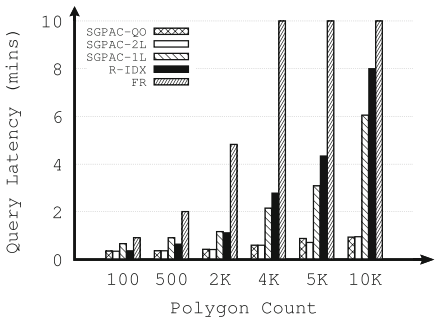
(c) Varying average polygon area (Grid Index)



(d) Varying average polygon area (RTree Index)



(e) Varying polygon count (Grid Index)



(f) Varying polygon count (RTree Index)

Fig. 5 Query performance on synthetically-composed polygon sets

7.3 Global index tuning

This section studies the effect of various global index settings on the query latency of all techniques. Names of approaches running on a grid local index are appended with (*G*), whereas names of approaches running on an R-Tree local index are appended with (*R*).

Figure 6 shows the effect of varying the partition capacity for quadtree global index on the four real polygon sets of Table 1. In Fig. 6(a) for continents and Fig. 6(b) for countries,

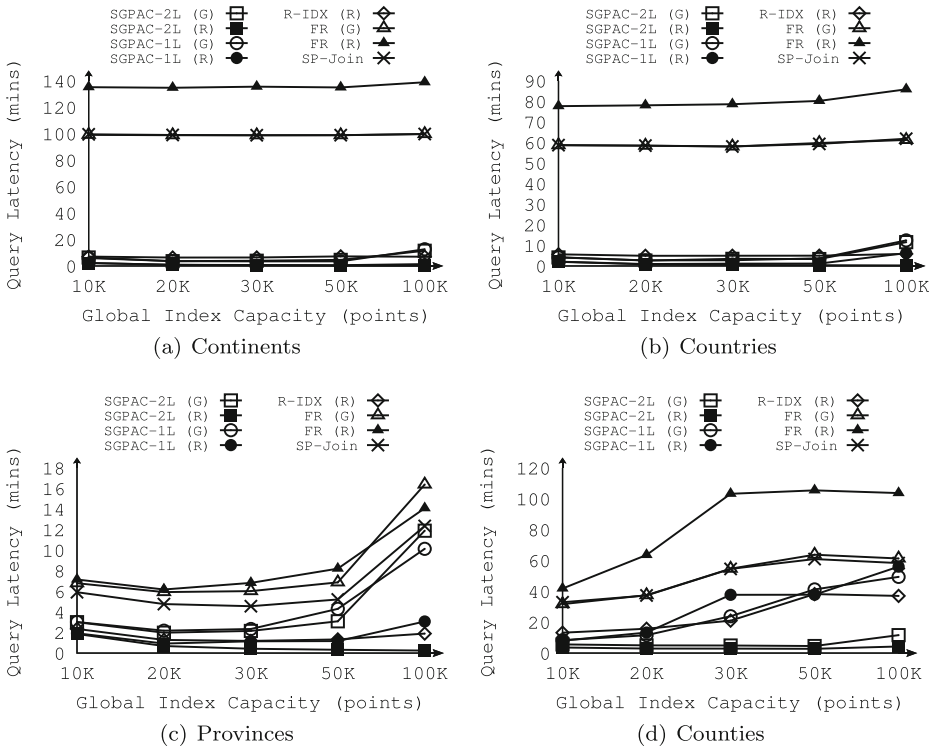


Fig. 6 Varying global index partition capacity on Real Polygon Layers

all techniques have almost stable query latency while varying the partition capacity. Moreover, all variations of *SGPAC* have significantly faster latency for all values compared to *FR* and *SP-Join* with up to 540 times faster when using an R-Tree local index for continents set, and up to 340 times faster with the same setup for the countries set. *SGPAC* variations have less speedup over *R-IDX* compared to *FR* and *SP-Join*, with faster 3–27 times faster query latency in case of continents and 3–24 times faster in case of countries.

Figure 6(c) and (d) show the performance for the provinces and counties polygon layers. Here we see an increasing latency pattern while increasing the partition capacity, where the increase is slight in the case of provinces and more significant in the case of counties. Provinces are relatively simple polygons with an average of 246 points per perimeter, yet they are larger in area than counties. As a result, increasing the partition capacity from 10K to 50K points does not significantly affect any of the techniques. For these simple polygons, all techniques using grid local index are comparable with the *SGPAC* latency up to only twice faster than *FR* and *SP-Join*. When using R-Tree index, *SGPAC* shows 2–7 times faster query latency than *R-IDX* and 4–60 times faster for *FR* when using an R-Tree index.

For counties, most of the polygons are of a small area, so they wholly fit in local partitions with almost no clipping at the global index level. With their complex perimeters, increasing the partition capacity significantly increases the point-in-polygon operations cost at each local partition and leads to higher latency. Yet, in this case, *SGPAC-2L* shows the best performance due to its effective perimeter complexity reduction at both global and local levels. *SGPAC-2L* has an order of magnitude faster latency than *FR* and *SP-Join*. *SGPAC-1L*

shows less performance improvement as it ignores the local clipping step. This again confirms our previous conclusions on the significant effectiveness of our two-level clipping as opposed to one-level clipping. *R-IDX* latency increases slightly with increasing the partition capacity, till it meets and outperforms *SGPAC-1L* as the partition capacity increases.

The same experiment has been repeated on synthetically-composed polygon sets Fig. 7 and produces similar results and the same conclusions. All *SGPAC* variations have significantly better performance than traditional techniques for different parameter values. In case of complex polygon set with larger areas as in Fig. 7(a), increasing the partition capacity does not have significant effect on the query latency. However in Fig. 7(c) where the average area is $13,000km^2$, all approaches latencies increase with increasing the partition capacity.

7.4 Local index tuning

This section studies the effect of varying local index granularity on the performance of *SGPAC-2L*. Figure 8 shows varying the cell size from 1KMx1KM to 100KMx100KM on the real polygon sets as well as the synthetically composed sets. For all real sets, changing the local grid granularity does not significantly affect query latency, except for the counties set. For counties, most of the polygons are of a small area, so they wholly fit in local partitions with almost no clipping at the global index level. If the local index cell size is large, the polygons could fit in one local partition skipping also the second clipping which in turn increases the running time significantly. For the synthetically-composed sets, we see that as

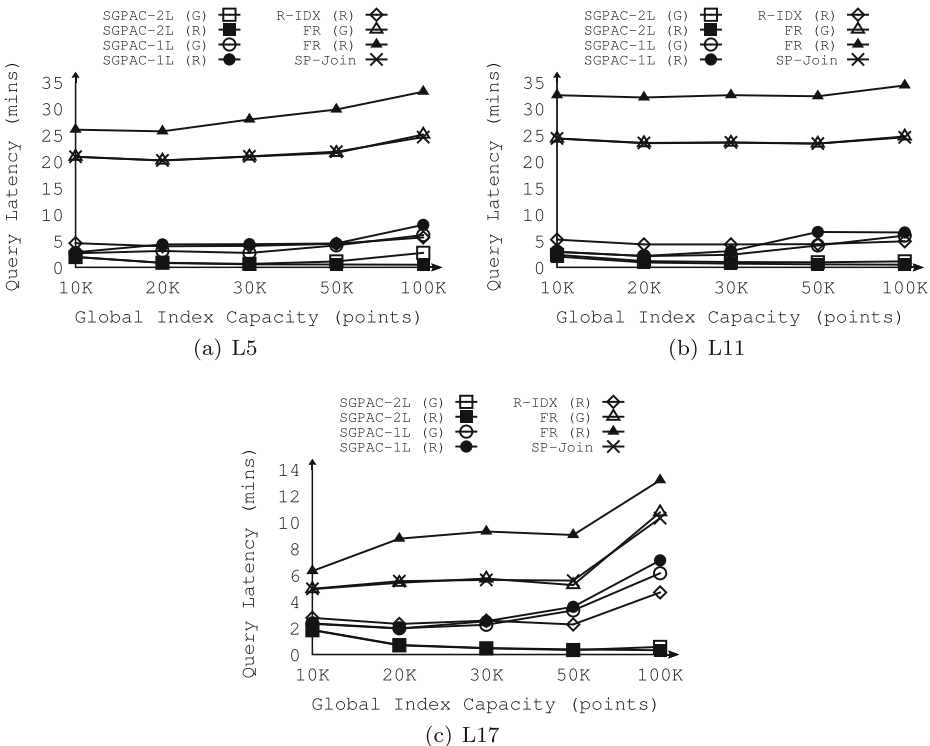


Fig. 7 Varying global index partition capacity on Synthetically-Composed Polygon Sets

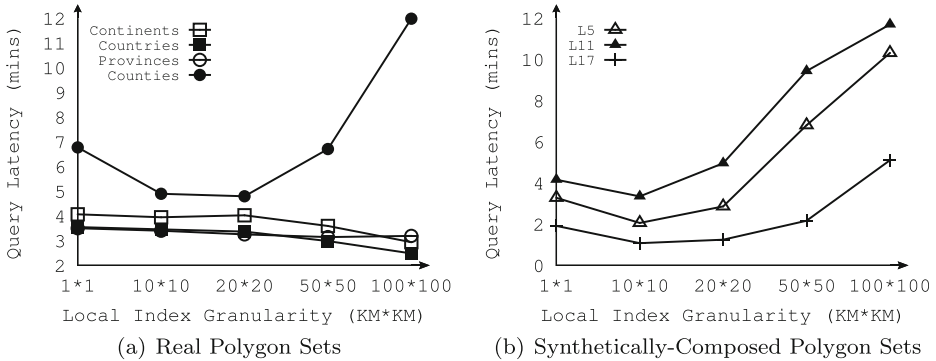


Fig. 8 Varying local grid index granularity

the cell size increases, the latency slightly increases because more polygons do not undergo sufficient clipping.

Figure 9 shows varying the R-Tree node capacity from 100 points per node to 10K points per node. Figure 9(b) shows that for Continents, Countries and Provinces sets the latency is not affected by changing the node capacity. However, as seen in Fig. 9(a), Counties

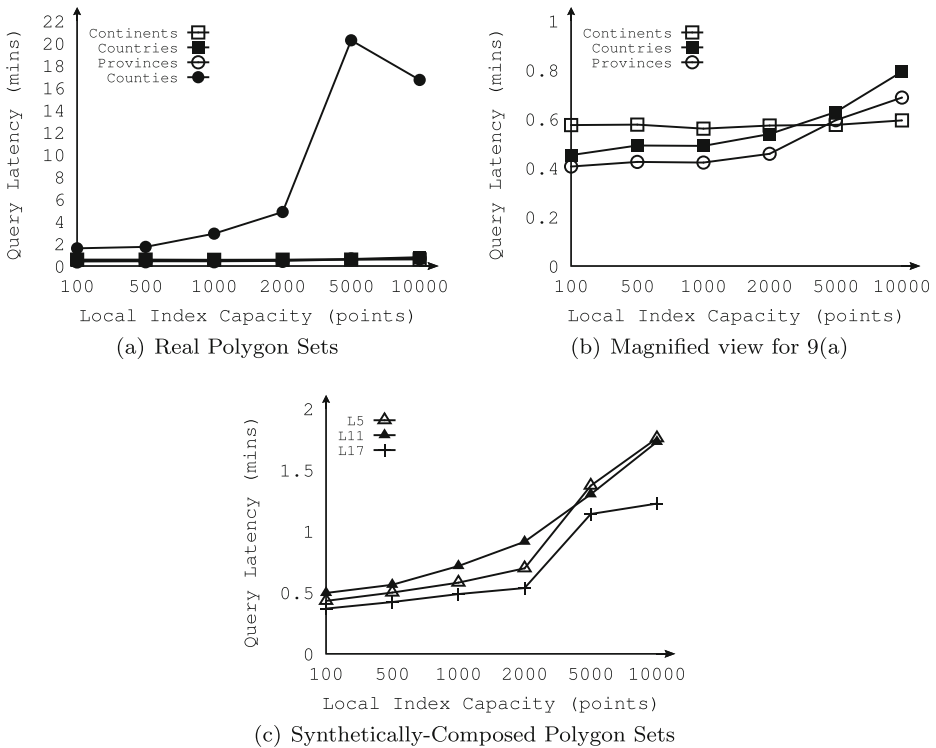


Fig. 9 Varying local R-Tree node capacity

sets latency significantly increases with increasing the node capacity. Similar to the Grid Index, increasing the R-Tree node capacity generates nodes with larger areas, which in turn does not reduce the polygon complexity sufficiently as smaller area nodes. Similar to the behaviour of synthetic sets using grid index, Fig. 9(c) shows that increasing the node capacity slightly causes an increase in the query latency.

7.5 Cost models evaluation

In this section, we first evaluate the accuracy of the cost models in the case of the local grid index introduced in Section 5. Then we evaluate the tree-to-grid mapping accuracy discussed in Section 6.2.

7.5.1 Grid index cost models evaluation

This experiment evaluates the accuracy of our cost models (that are presented in Section 5) in terms of the number of basic operations, rather than query latency as discussed in previous experiments. Figure 10 shows the estimated number of operations for the models in (1) (denoted as *Estimated FR*) and (2) (denoted as *Estimated SGPAC-2L*) versus the actual number of operations (denoted as *Actual FR* and *Actual SGPAC-2L*, respectively). The figure clearly shows the perfect match of *Estimated FR* and *Actual FR*, so the filter-refine cost model calculates the actual processing cost very accurately. However, *Estimated SGPAC-2L* is much higher than *Actual SGPAC-2L* in all cases except counties. This means *SGPAC* cost model in (2) always overestimates the cost. The reason is that this model assumes a very conservative assumption that *all* overlapping grid cells lie on the polygon borders and perform refine operations. This assumption is so conservative for sufficiently large polygons for most grid cell sizes. Many of these grid cells lie completely inside the polygon and do not perform any refine operations. This assumption is more realistic for small polygons such as counties, and that is why Fig. 10(e) shows the most accurate estimations for this model. This assumption is also affected by the grid cell size. The larger the grid cell, the fewer cells are wholly contained in the polygon, the more accurate the model. So, in all figures, when grid cell size increases, the model estimates more accurately. In all cases, it is noticeable that the model always provides an upper bound for the actual cost and never underestimates it.

7.5.2 Tree-to-grid evaluation

As discussed in Section 6.2, even in the case of a local tree index, we maintain a grid estimator. The main challenge is choosing a grid cell granularity that is small enough to provide accurate estimations but not too small, to be an overhead on the system to maintain. Figure 11 shows the ratio between the expected number of operations to the actual number of operations while changing the number of cells in each local estimator grid. For the continents and the countries' datasets, the grid cell size does not affect the estimation accuracy. The reason is that these polygons are large enough to cover the whole local partition. Hence dividing the local partitions into more cells does not significantly affect the estimation outcome. On the other hand, the provinces and the counties datasets show estimation accuracy improvement while increasing the number of cells per partition. This observation is more evident in the case of the counties dataset since these polygons have the smallest area.

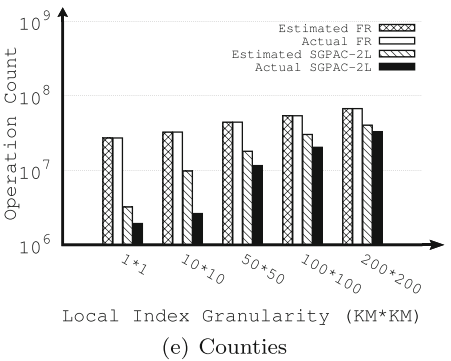
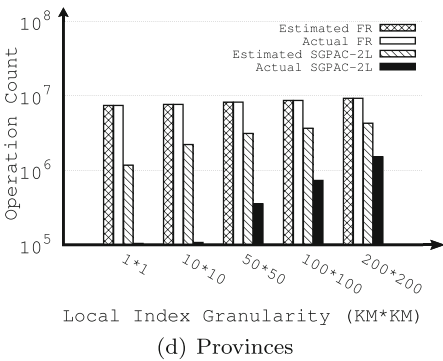
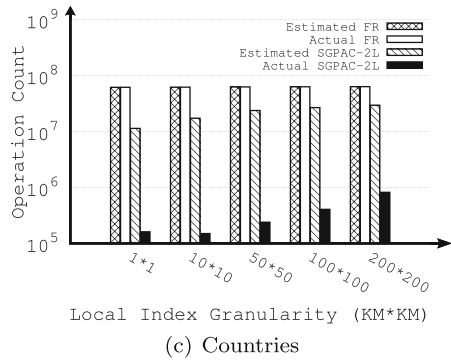
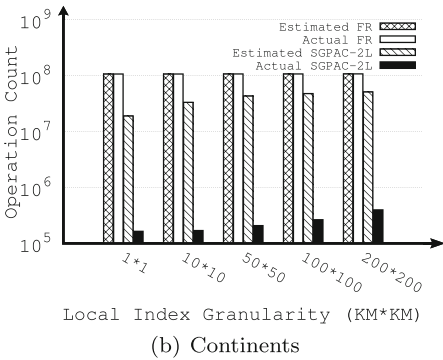
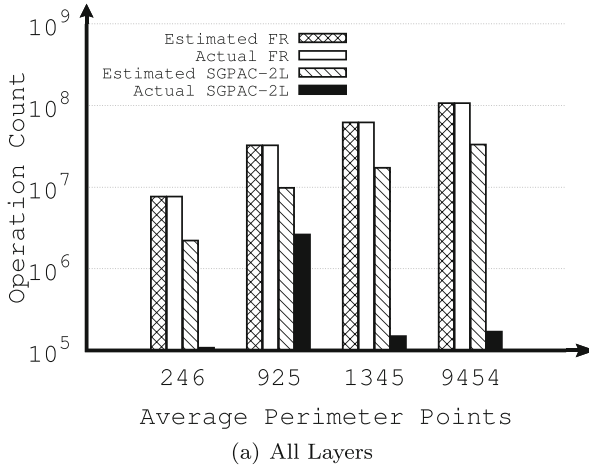


Fig. 10 Grid index cost models evaluation

8 Conclusions

This paper proposes a spatial group-by query that groups spatial points based on the boundaries of a set of complex polygons. This query designed for thousands of polygons with complex perimeters and large-scale spatial datasets that contain hundreds of millions of

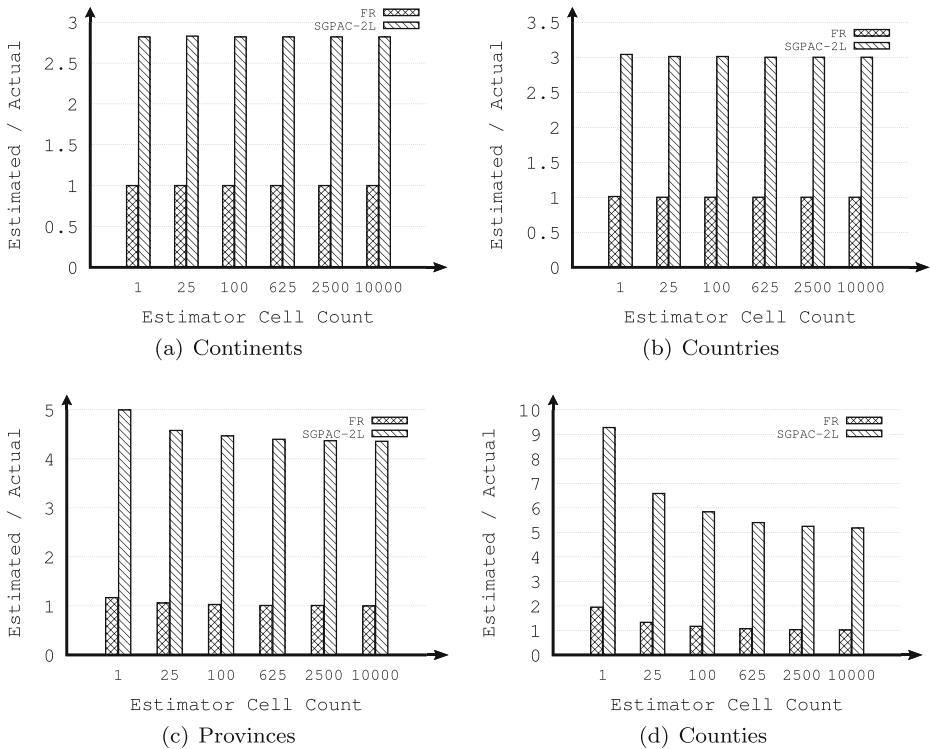


Fig. 11 Tree-to-grid estimator evaluation

points. We have proposed a highly efficient query processor that uses existing distributed spatial data systems to support scalable spatial group-by queries. Our technique depends on reducing query polygons complexity by clipping them based on global and local spatial indexes. Furthermore, we distinguish highly skewed data partitions to enforce load balancing among distributed workers. In addition, we employ an effective query optimizer that adaptively assigns an appropriate processing scheme based on polygon complexity. The experimental evaluation showed significant superiority for our techniques over existing competitors to support large-scale datasets with up to an order of magnitude faster query latency.

Acknowledgements This work was partially supported by the National Science Foundation, under grants IIS-1849971, IIS-1901379 and SES-1831615 is also their COI.

Data Availability All data created or used during this study are publicly available at the following web-sites: <https://star.cs.ucr.edu/>, <https://www.natureearthdata.com/>, <https://www.arcgis.com/>, and <https://gadm.org/data.html>.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Abdelhafeez L, Magdy A, Tsostras VJ (2020) Scalable spatial GroupBy aggregations over complex polygons. In: SIGSPATIAL
2. Aghajarian D, Puri S, Prasad S (2016) GCMF: an efficient end-to-end spatial join system over large polygonal datasets on GPGPU platform. In: SIGSPATIAL
3. Almaslukh A, Magdy A, Rey SJ (2018) Spatio-temporal analysis of meta-data semantics of market shares over large public geosocial media data. *J Locat Based Serv*
4. Azevedo LG, Güting RH, Rodrigues RB, Zimbrão G, de Souza JM (2006) Filtering with raster signatures. In: ACM GIS
5. Azevedo LG, Zimbrão G, De Souza JM (2007) Approximate query processing in spatial databases using raster signatures. In: *Advances in geoinformatics*. Springer
6. Brinkhoff T, Kriegel H-P, Schneider R (1993) Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems. In: ICDE
7. Brinkhoff T, Kriegel H-P, Schneider R, Seeger B (1994) Multi-step processing of spatial joins. *SIGMOD* 23(2)
8. (2017) ArcGIS Continents. <https://www.arcgis.com/home/item.html?id=5cf4f223c4a642eb9aa7ae1216a04372>. Accessed 2019
9. Cortes RX, Rey S, Knaap E, Wolf LJ (2019) An open-source framework for non-spatial and spatial segregation measures: the PySAL segregation module. *J Comput Soc Sci*
10. GADM Counties. https://gadm.org/download_world.html. Accessed 2019
11. (2009) NE Countries. <https://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-adm-in-0-countries/>. Accessed 2019
12. (2009) NE Provinces. <https://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-adm-in-1-states-provinces/>. Accessed 2019
13. Duque JC, Anselin L, Rey SJ (2012) The Max-p-regions problem. *J. Reg Sci* 52(3)
14. Spatialhadoop EA, Mokbel MF (2015) A Mapreduce framework for spatial data. In: ICDE
15. Eldawy A, Mokbel MF (2015) The era of big spatial data. In: ICDEW
16. Fang Y, Friedman M, Nair G, Rys M, Schmid A-E (2008) Spatial indexing in microsoft SQL server 2008. In: *SIGMOD*
17. Frank AU (1981) Application of DBMS to land information systems. In: VLDB
18. Guo Q, Palanisamy B, Karimi HA, Zhang L (2016) Distributed algorithms for spatial retrieval queries in geospatial analysis. *STCC* 4(3)
19. Güting RH (1994) An introduction to spatial database systems. *VLDB J*
20. Haines E (1994) Point in polygon strategies. *Graphics gems IV* 994
21. Hu Y, Ravada S, Anderson R (2011) Geodetic point-in-polygon query processing In Oracle spatial. In: *SSTD*
22. Hu Y, Ravada S, Anderson R, Bamba B (2012) Topological relationship query processing for complex regions In oracle spatial. In: *GIS*
23. Hughes JN, Annex A, Eichelberger CN, Fox A, Hulbert A, Ronquest M (2015) GeoMesa: a distributed architecture for spatio-temporal fusion. In: *SPIE*, vol 9473
24. Jacox EH, Samet H (2007) Spatial join techniques. *TODS* 32(1)
25. Kang W, Rey S, Wolf L, Knaap E, Han S (2020) Sensitivity of sequence methods in the study of neighborhood change in the United States. *Comput Environ Urban Syst* 81
26. Kipf A, Lang H, Pandey V, Persa RA, Anneser C, Zacharatou ET, Doraiswamy H, Boncz PA, Neumann T, Kemper A (2020) Adaptive main-memory indexing for high-performance point-polygon joins. In: *EDBT*
27. Kothuri RK, Ravada S (2001) Efficient processing of large spatial queries using interior approximations. In: *SSTD*
28. Kothuri RKV, Ravada S, Abugov D (2002) Quadtree and R-tree indexes in oracle spatial a comparison using GIS data. In: *SIGMOD*
29. Kriegel H-P, Horn H, Schiwietz M (1991) The performance of object decomposition techniques for spatial query processing. In: *SSTD*
30. Lee Y-J, Lee D-M, Ryu S-J, Chung C-W (1996) Controlled decomposition strategy for complex spatial objects. In: *DEXA*
31. Lee Y-J, Park H-H, Hong N-H, Chung C-W (1996) Spatial query processing using object decomposition method. In: *CIKM*
32. Liu R, Zhang H, Busby J (2008) Convex hull covering of polygonal scenes for accurate collision detection in games. In: *Graphics interface*
33. Malensek M, Pallickara S, Pallickara S (2013) Polygon-based query evaluation over geospatial data using distributed hash tables. In: *UCC*

34. Nodarakis N, Sioutas S, Gerolymatos P, Tsakalidis A, Tzimas G (2015) Convex polygon planar range queries on the cloud: grid vs. angle-based partitioning. In: ALGOCLOUD
35. Orenstein JA (1989) Redundancy in spatial databases. SIGMOD 18(2)
36. Pandey V, Kipf A, Neumann T, Kemper A (2018) How good are modern spatial analytics systems? VLDB 11(11)
37. Ray S, Simion B, Brown AD, Johnson R (2014) Skew-resistant parallel in-memory spatial join. In: SSDBM
38. Rey SJ, Anselin L (2007) PySAL: a python library of spatial analytical methods. Rev Reg Stud 37(1)
39. Shamos MI, Hoey D (1976) Geometric intersection problems. In: SFCS
40. Sidlauskas D, Chester S, Zacharatos ET, Ailamaki A (2018) Improving spatial data processing by clipping minimum bounding boxes. In: ICDE
41. Toth CD, O'Rourke J, Goodman JE (2017) Handbook of Discrete and Computational Geometry Chapman and Hall
42. (2020) UCR STAR. <https://star.cs.ucr.edu/>. Accessed 2019
43. (2020) UCR STAR NE Provinces. https://star.cs.ucr.edu/?NE/states_provinces#center=34.0,-117.3&zoom=2. Accessed 2019
44. Wang N, Hu B-G (2011) IdiotPencil: an interactive system for generating pencil drawings from 3D polygonal models. In: CADGRAPHICS
45. Xie D, Li F, Yao B, Li G, Zhou L, Guo M (2016) Simba: Efficient in-memory spatial Analytics. In: SIGMOD
46. Yu J, Zhang Z, Sarwat M (2018) Spatial data management in apache Spark: the GeoSpark perspective and beyond. Geoinformatica
47. Zacharatos ET, Doraiswamy H, Ailamaki A, Silva CT, Freiref J (2017) GPU rasterization for real-time spatial aggregation over arbitrary polygons. VLDB 11(3)
48. Zhang D, Tsotras VJ (2001) Improving min/max aggregation over spatial objects. In: Proceedings of the 9th ACM international symposium on advances in geographic information systems, GIS '01. Association for Computing Machinery, New York, pp 88–93
49. Zhang J, You S (2012) Speeding up large-scale point-in-polygon test based spatial join on GPUs. In: SIGSPATIAL
50. Zimbrão G, De Souza JM (1998) A raster approximation for processing of spatial joins. In: VLDB

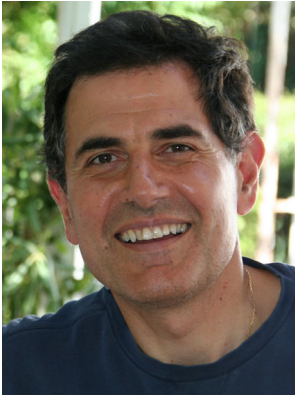
Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Laila Abdelhafeez received her B.Sc. in Computer Engineering, from faculty of Engineering, Cairo University, Egypt in 2017. She is currently pursuing her Ph.D. degree in Computer Science at the University of California, Riverside. Her research interests include big data management, and spatial data management.



Amr Magdy is an Assistant Professor of Computer Science and Engineering and a cofounding faculty member of the Center for Geospatial Sciences at UC Riverside. His research interests include database systems, spatial data management, big data management, largescale data analytics, indexing, and main-memory management. His research is published in prestigious research venues, including ACM SIGMOD, ACM SIGSPATIAL, IEEE ICDE, IEEE TKDE, and VLDB Journal. Amr's research is recognized among the best papers in IEEE ICDE 2014 and ACM SIGSPATIAL 2019 and has been incubated by several industrial collaborators. He is a recipient of the NSF CRII 2019 award and several research grants that support his research on spatial data science and related educational activities.



Vassilis J. Tsotras is a professor at the Department of Computer Science and Engineering at UC Riverside (UCR). Before that he was a (tenured) Associate Professor at Polytechnic University (now New York University). Currently, he serves as the director of the UCR Data Science Center and Chair of the UCR Data Science program. His research is in the general area of Databases with focus on big data management and spatial and spatio-temporal data. He serves as editor in chief of the Journal of Cooperative Information Systems and have served as associate editor for the IEEE Transactions on Knowledge and Data Engineering (TKDE) and The Very Large Data Bases (VLDB) Journal. He was the General co-Chair of the 26th International Conference of Data Engineering (ICDE) and Program Chair (DB Track) for the 15th Conference on Information and Knowledge Management (CIKM). His research has been supported through various grants from NSF, the Department of Defense and industry.