Geometry and Proximity Constrained Query Evaluations over Large Geospatial Datasets Using Distributed Hash Tables

Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara, *Members, IEEE*

Abstract

Data volumes in the geosciences and related domains have grown significantly as sensing equipment designed to continuously gather readings and produce data streams for geographic regions have proliferated. The storage requirements imposed by these datasets vastly outstrip the capabilities of a single computing resource, leading to the use and development of distributed storage frameworks composed of commodity hardware.

In this paper, we explore the challenges associated with supporting geospatial retrievals constrained by arbitrary geometric bounds, geographic proximity, and relevance rankings. Our solution involves the use of a lightweight, distributed spatial indexing structure, the *geoavailability grid*. Geoavailability grids provide global, coarse-grained representations of the spatial information stored within these ever-expanding datasets, allowing the search space of distributed queries to be reduced by eliminating storage resources that do not hold relevant information. The index can also be used in non-distributed settings, and performs competitively with other spatial indexing technologies.

Index Terms

Geospatial Query Evaluation, Distributed Hash Tables, Cloud Infrastructure

1 Introduction

The proliferation of observational devices such as in situ sensors and remote sensing equipment such as satellites and radars have contributed to ever-increasing data volumes. These sensors measure and report on various environmental and atmospheric phenomena that are used in weather forecasting, ecology, hydrology, erosion, and agricultural models. The rate, resolution, and precision at which these measurements are performed have all increased over time, leading to the collection of extreme-scale datasets that logically fuse information gathered from diverse equipment.

To cope with these data volumes and their concomitant I/O loads, such datasets are dispersed over a collection of machines for future analysis and retrieval. We investigate this problem in the context of distributed hash tables (DHTs). DHTs are robust, scalable systems for managing large networks of heterogeneous computing resources. Often underpinned by a consistent hashing scheme, DHTs offer excellent load balancing properties and are well-suited for scale-out architectures where commodity hardware can be added incrementally to meet rising storage or processing demands.

Analysis of such datasets often involves queries on spatial bounds of interest in the form of user-specified geometric shapes. Such queries can correspond to administrative or natural boundaries, and provide greater freedom to apply various types of investigation or processing. Storage and retrieval of predefined polygon-based shapes is well researched; a typical approach involves sorting polygon coordinates along one dimension (such as latitude or longitude) to generate a deterministic array that can be used to compute a hash value for storage and retrieval. However, the datasets we focus on in this work are multidimensional and continually assimilate additional data at varying resolutions from diverse sources. This renders solutions that rely on precomputed or static shapes ineffective, necessitating an alternate approach. Consequently, the geometry-based query support in question must be decoupled from the generation and storage of data.

1.1 Research Challenges

We consider the problem of fast and scalable evaluation of queries constrained by arbitrary shapes over time series datasets with geospatial properties. The challenges involved in doing so include:

• *M. Malensek, S. Pallickara, and S. Pallickara are with the Department of Computer Science, Colorado State University, Fort Collins, CO 80523. E-mail: {malensek,sangmi,shrideep}@cs.colostate.edu*

1) The data being managed is both voluminous and distributed over multiple computing resources.
2) The system is decentralized; distributed query evaluations can be performed by any of the machines that comprise the storage network.
3) Broadcasting to all machines for query evaluation is inefficient and latency-prone; the search space must be reduced to efficiently service query requests and facilitate timely analysis.
4) Data points have multiple dimensions that represent a variety of readings for a particular geolocation.
5) Queries may specify chronological bounds to request a portion of the available time series information.
6) Distributed data structures used for query evaluation must be compact to avoid excessive state exchange.

1.2 Research Questions

Key research questions that we explore in this paper include the following:

1) How can we manage the trade-off space between memory consumption and the resolution of data structures? How does this impact the speed of query evaluations?
2) How do we strike a balance between global and local information maintained by each node, and what is the impact on the overall search space?
3) How can we constrain query evaluations using arbitrary shapes without compromising key DHT functionality?
4) How can we support proximity queries efficiently? In some cases, query scope may be controlled by requirements for a specific number of results.
5) Given the high dimensionality of the data points, how can we support efficient relevance ranking of query evaluation results?

1.3 Overview of Approach

The approach described in this paper is based on our hierarchical DHT implementation, Galileo [1], [2]. Galileo is designed for high-throughput management of multidimensional data streams. To create an overall view of the spatial locations of data stored in the system, we provide a distributed spatial index called the ***geoavailability grid***. Updates to the grid are disseminated through a lightweight gossip protocol, and we rely on an eventual consistency model wherein nodes in the system will converge on a steady state when no new updates are available.

This work extends our previous investigation [2] on the distributed lookup capabilities that can be provided by this indexing technology: hybrid local retrievals (Section 5), arbitrary geometric shapes or combinations of shapes that need not be contiguous (Section 6), proximity searches (Section 7), and relevance rankings (Section 8).

1.4 Paper Contributions

A key innovation in the algorithms described in this paper is the ability to constrain search queries using arbitrary shapes (including curves and polygons) in multidimensional, spatiotemporal datasets encompassing hundreds of millions of files. Aspects of our algorithm are also amenable to GPU acceleration. To our knowledge, no current system provides the real-time query evaluation capabilities described in this paper.

Additionally, our framework accelerates queries by evaluating them concurrently across relevant nodes while supporting expressive range-based and exact-match retrievals on feature values in addition to polygon boundaries. Most importantly, data generation and query specification are completely decoupled in our solution, and do not rely on any of the previously-developed functionality in Galileo. This makes the Geoavailability Grid applicable to other storage frameworks as well.

To ensure the scalability of our solution we have also conducted preliminary tests that doubled the size of the dataset and increased the number of nodes in the cluster by 60%, while achieving a similar performance profile in both query response times and communication per node. Results from a simulated run on a 10,000-node cluster suggest that these trends continue at larger scales.

2 System Overview

Galileo is a high-throughput storage framework implemented as a distributed hash table (DHT). Unlike typical DHT designs, Galileo uses *zero-hop* routing, meaning requests are sent directly to their destination rather than taking intermediate hops through the network. This reduces latency and the overall amount of communication between network participants. To ensure scalability and flexibility, Galileo also supports the use of multiple hash functions to create resource hierarchies or subdivide the network. The system has been tested with up to 10,000 nodes in a simulated environment and been deployed in production on clusters ranging from 48 to 150 nodes.

The primary use case for Galileo is the storage and processing of voluminous, multidimensional datasets in the scientific domain. These datasets often have spatial and temporal characteristics along with several other *features*

of interest. For retrievals based on feature values, the system allows both exact-match and range-based queries through a multi-layered indexing strategy that incorporates a global *feature graph* and local *metadata graph* instances. To facilitate processing activities through MapReduce computations or directed, cyclic graphs, Galileo is integrated with the Granules cloud runtime. Granules has been used to process data streams in settings with stringent real time constraints [3].

Unlike a standard DHT, partitioning and retrieval operations in Galileo are completely decoupled. This approach allows for novel load distribution configurations and hierarchical network layouts. The *storage nodes* in Galileo can be placed into *groups* to create a network hierarchy; in this study, each group is assigned a portion of the overall geography being managed by the system.

2.1 Metadata and Information Retrieval

Each node in the system maintains a *metadata graph* for quickly evaluating local queries. A metadata graph instance is populated with relevant feature information from the files stored on the node, and traversing through the graph's hierarchical structure narrows queries down to their relevant files. For global lookup capabilities, a *feature graph* instance is maintained at each storage node, which provides a coarse-grained view of all the data in the system. When executing a distributed query, the feature graph can be used to reduce the overall search space before submitting individual subqueries to local metadata graphs for evaluation. In most cases, this optimization provides a dramatic reduction in the number of nodes that must be contacted to evaluate a given query operation.

While the combination of the metadata graphs and feature graph have proven to be effective for indexing a variety of data types, storing two- or even three-dimensional information in these structures can involve the creation of a substantial amount of vertices and edges. For instance, the high-resolution data subset used in this work required the addition of at least 33 million vertices per graph under optimal conditions. These factors, as well as the analysis capabilities that spatial indexing can afford, inspired our development of the geoavailability grid.

2.2 Experimental Configuration

Our test dataset was derived from the National Oceanic and Atmospheric Administration (NOAA) North American Mesoscale Forecast System [4]. The test dataset consists of one billion (1,000,000,000) files, each around 8 KB. The features that we indexed for this work included the spatial location of the samples, the time they were recorded, percent maximum relative humidity, surface temperature (Kelvin), wind speed (meters per second), and snow depth (meters).

Tests in this paper were carried out on a 48-node cluster of HP DL160 servers equipped with a Xeon E5620 CPU, 12 GB of RAM, and a 15000-RPM disk. Nodes were divided into eight Galileo groups.

3 Indexing: The Geoavailability Grid

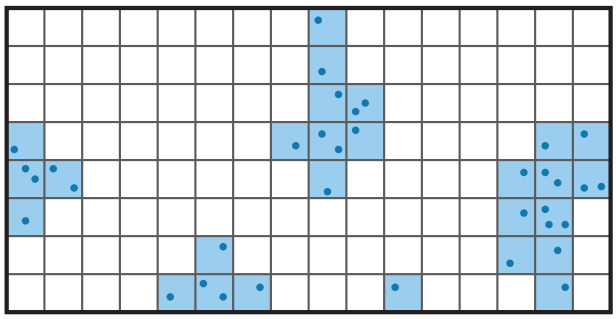
Indexing in a distributed environment with highly voluminous datasets can be challenging; a central "index" server is a single point of failure and can quickly become a bottleneck in high-load situations, but an index that is shared across all nodes in the system can result in consistency problems and excessive state exchange over the network.

The R-tree [5] is commonly employed in non-distributed applications for spatial indexing due to its speed and efficiency, but has several constraining properties that limit its scalability in distributed applications. Using an R-tree as a global index for billions of files would require a substantial amount of memory, along with a high number of distributed updates due to the frequent rebalancing operations that take place within the tree. Additionally, splitting the tree across multiple nodes and designing a storage network around the data structure is constraining and latency-prone.

To overcome these scalability issues, we have developed the *geoavailability grid*, a distributed spatial indexing data structure that is scalable and fault-tolerant. Geoavailability grids translate points in space to a reduced-resolution coordinate system for indexing purposes using bit vectors (bitmaps). Each bit represents a location, and its on-off state indicates whether or not information has been stored there. Due to their concise and efficient nature, bitmap indexes have seen considerable research and usage in a variety of relational databases and data warehousing systems.

3.1 Geocoding

To partition information in our DHT and provide a coarse-grained representation of its spatial properties, we use the Geohash [6] geocoding algorithm. Geohash provides a hierarchical, grid-based model of the Earth where locations are represented by Base32 strings. The longer the Geohash string, the more precise the bounding box around the location it references. A Geohash is derived by interleaving bits obtained from latitude-longitude pairs; for example,



0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	1	0	0	0	0	1	1
1	1	0	0	0	0	0	0	1	0	0	0	0	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0

Fig. 1. A geographic region (left) containing several data points, with its geoavailability grid (right).

the decimal coordinates of N $41.8827°$, W $-87.6236°$ would map to the Geohash string *DP3WQ0D2*, representing 40 bits of precision (eight characters, five bits per character). Each additional bit in a Geohash doubles the number of hash buckets it references, representing finer-grained spatial areas that lie deeper within the hierarchy.

Using a geocoding algorithm is an essential component of our indexing scheme because it determines the ranges of information that must be stored in each instance of the index. In this study, the first two Geohash characters of a spatial location are used to determine the group of nodes responsible for storing the data. This has two key benefits: specifying the first two characters (10 bits) of a Geohash can significantly reduce the search space for spatial queries without additional indexing, and it also means that nodes can exclude information from their geoavailability grids that lies outside of their geographic scope.

3.2 Generating the Index

Geoavailability grids are initially configured with a width and height based on geocoding granularity. For example, if a gridded dataset contains readings at intervals of around 30 km, approximately 32 bits of Geohash precision would be required to place samples in separate "bins." Choosing an appropriate granularity is highly dependent on the type of information being stored and the intended analysis that will be performed on the data. Finer-grained resolutions allow more specific queries to be resolved, but also increase the overall size of the index. Each feature of interest (such as humidity or temperature) is accompanied by a unique geoavailability grid, enabling queries to distinguish between different feature types.

For our particular dataset, spatial locations are represented by 30-bit Geohashes. After accounting for the first 10 bits that are used to determine group membership, the remaining 20 bits are used to populate the geoavailability grids on each node in the system:

$$DP3WQ0 = \underbrace{01100\ 10101}_{\text{Group Hash}}\ \underbrace{00011\ 11100\ 10110\ 00000}_{\text{Location in Bitmap Index}}$$

This is accomplished by mapping spatial coordinates to their closest bitmap coordinates, and ensuring that the relevant bitmap location is set to a **1** to indicate that one or more data points are present in the location. 20 bits of precision corresponds to 2^{20} Geohash buckets, which is the total number of bits in each index instance. Since Geohashes interleave latitude and longitude values, the width and height proportion of the index changes with each additional bit. Therefore, an index of n Geohash bits would have a width of $2^{\lfloor n/2 \rfloor}$ and a height of $2^{\lceil n/2 \rceil}$. Figure 1 illustrates how a region could be represented as a geoavailability grid.

3.3 Compression

While bitmaps provide a simple means to index a wide variety of data types, the sheer number of bits required for these representations can prove to be problematic both in memory consumption and processing times for bitwise operations. Extensive investigation has been conducted on compressing bitmap representations, from simple run-length encoding to more advanced schemes such as the Enhanced Word-Aligned Hybrid (EWAH) compression [7], [8], which is used in this work. Table 1 illustrates the difference between uncompressed and compressed bitmap representations for our **entire** dataset. For gridded data, higher resolutions (derived from the second half of the Geohash bits) increase the sparsity of the index and improve compressibility. If our dataset was expanded to cover the entire Earth, the total size of the geoavailability grids would be about 50 MB using 25-bit precision.

Compressed bitmaps are somewhat unique in that they generally do not require decompression before processing occurs. In fact, compression can often speed up bitwise operations. To deal with situations that require alternative compression algorithms, we provide an interface that allows the underlying bitmap representation of a geoavailability grid to be changed at runtime or during system configuration.

TABLE 1
Bitmap Compression for Various Index Resolutions

Resolution	Original Size (KB)	Compressed (KB)
15-bit	309.0	294.4
20-bit	9879.02	3196.9
25-bit	316090.28	4034.7

3.4 Updating the Index

To ensure that new files' spatial information is disseminated rapidly, geoavailability grid updates are gossiped between groups on a regular basis along with other state information. An update consists of a set of bits that have changed since the publication of the previous update. If a storage node finds itself out of sync with the current updates, neighboring peers can also generate an update or transmit an entire copy of the index. Updates are also represented as compressed bitmaps, meaning that they generally consume a minimal amount of space; in our tests, a completely random 1000-bit update (representing an approximate worst case from a compressibility standpoint) resulted in an update size of about 15 KB.

4 Retrieval: Polygon-Based Query Evaluation

Once the spatial information has been indexed in geoavailability grids at each storage node, the system can evaluate user-defined geospatial queries. Geospatial query evaluation in Galileo proceeds as follows:

1) A user submits query geometry to retrieve data from.
2) The query is *decomposed* into subqueries by intersecting it with group geometries.
3) Geoavailability grids are consulted to determine if data may be available, eliminating any irrelevant nodes.
4) Subqueries are submitted to the remaining set of relevant nodes for evaluation.

4.1 Spatial Decomposition

Each group in Galileo is responsible for storing data pertaining to a particular geospatial region. These regions are known by the other nodes in the system and maintained in memory as polygons. To begin decomposing a spatial query, the minimum bounding rectangle (MBR) is calculated for the query geometry, which is the smallest rectangle that completely surrounds the query polygon. Any group geometries that are overlapped by the query MBR are then intersected with the query polygon. After the intersection operation, the remaining geometries are used to produce a set of groups that are relevant to the query. Decomposing queries in this manner has two key advantages: small queries will naturally involve fewer storage nodes, whereas larger queries that are represented by polygons spanning greater geographic regions are processed in parallel across multiple nodes.

4.2 Geoavailability Evaluation

Before being evaluated against the collection of pertinent geoavailability grids, query polygons must be projected onto a corresponding bitmap coordinate system. Once this process has been completed, a *query bitmap* is created using the polygon geometry. To create a query bitmap, the spatial area covered by the geoavailability grid can be thought of as a monochrome graphical canvas that will be drawn using standard graphics routines; using the provided query polygon, the regions of interest are filled with color to set the relevant bits within the polygon boundaries to **1**. This effectively converts a user-provided polygon into a geoavailability grid by leveraging existing graphical algorithms and any hardware acceleration available to the system. In cases where extremely large bitmaps are being generated, GPUs can be leveraged if support is available. Another benefit of this strategy is that queries can be easily visualized as images.

Once a query bitmap has been obtained, evaluating the presence of relevant data within the polygon boundaries is simple: a logical AND is performed between the geoavailability grids and query bitmap. If the resulting bitmap contains *any* bits set to **1**, then there was a region with relevant spatial data that overlapped the query geometry, and the subquery is passed on to relevant storage nodes. Table 2 contains timing information for processing a geoavailability lookup. We also performed the same test with MongoDB 2.4 [9] by using the polygon-based `$geoWithin` operator to query against our dataset; the evaluations took 28.42ms on average, with a standard deviation of 1.51ms. However, the spatial index used in MongoDB does not reduce the resolution of its data points.

A complete geoavailability evaluation returns a set of storage nodes that contain spatial information within the query boundaries. This set is intersected with results from a feature graph lookup of any other constraints specified

TABLE 2
Geoavailability evaluation speed, averaged over 1000 runs against each group Geohash

Resolution	Lookup Time (ms)	SD (ms)
15-bit	0.012	0.021
20-bit	0.163	0.203
25-bit	0.723	0.289

by the user, which further reduces the search space by eliminating any destinations that cannot satisfy the entire query. *Subqueries* are submitted to the remaining set of storage nodes for the final step in the query evaluation process: local retrieval.

5 Local Retrieval: Geoavailability R-trees

While our previous investigations showed that geoavailability grids can outperform the venerable R-tree spatial index at extreme scales [2], the grids benefit from using coarse-grained spatial representations provided by geocoding. To take advantage of the strengths present in both technologies, we developed a hybrid form of R-tree that uses geospatial boundaries derived from geoavailability grids. In this case the R-tree contains geoavailability grid cells rather than individual points, which decreases precision but significantly reduces the amount of information that must be stored in the index.

To investigate the performance of the geoavailability R-tree, geoavailability grid, and a standard R-tree, we used a polygon subquery in North America from our previous work that followed the Mississippi river down to the Gulf of Mexico. The Java Spatial Index (JSI) [10] implementation of the R-tree algorithm was used in these benchmarks due to its focus on performance, and the geoavailability grids were configured with 25-bit precision. Through the course of the tests, the query polygon was upscaled to retrieve more files by including a larger portion of the dataset. Figure 2 illustrates the performance of all three strategies when retrieving points from our test dataset, with each point representing about 5000 files. While the standard R-tree cannot cope with large amounts of data points, the geoavailability R-tree provides the best overall performance in our benchmarks. However, using a geoavailability R-tree requires additional memory for each feature type, and also exhibits slower performance as the number of points retrieved increases; evaluating a query with a geoavailability grid will generally require the same amount of processing time regardless of the size of the query geometry.

6 Advanced Query Geometry

A broad range of geometric concepts can be represented with polygons alone. However, polygons are not always the most concise way to portray a geospatial query. For instance, circular shapes that can be described with an origin and radius would require a large amount of very small line segments to be approximated with a polygon. Describing a square or rectangle requires only two opposite coordinate pairs, versus four line segments in an equivalent polygon. For this reason, we allow several additional geometry types to be used in spatial queries to a handle a more comprehensive range of use cases.

When dealing with complicated geometry, geoavailability grids provide results that require very little post processing, unlike indexes such as R-trees that generalize shapes to rectangles. A single query may also include multiple shapes that are combined with operators from set notation: for example, a union of two shapes would select the geographical regions encompassed by both shapes, whereas the complement operator can be used to effectively "subtract" regions from an existing shape.

6.1 Shape Types and Transformations

We provide several types of shape geometry beyond standard polygons for expressing queries:

- Line segments
- Quadratic and cubic (Bézier) curves
- Squares, rectangles, and rounded rectangles
- Circles and ellipses

Each of these shape types can be combined with set operators or be modified through a *shape transformation*. Transformations include translation, rotation, scaling, clipping, and shearing, which can be useful for adapating simple geometric shapes to a particular terrain. Queries can be created programmatically or generated with any graphics software capable of creating SVG images, which can simplify the query process for end users. Once a

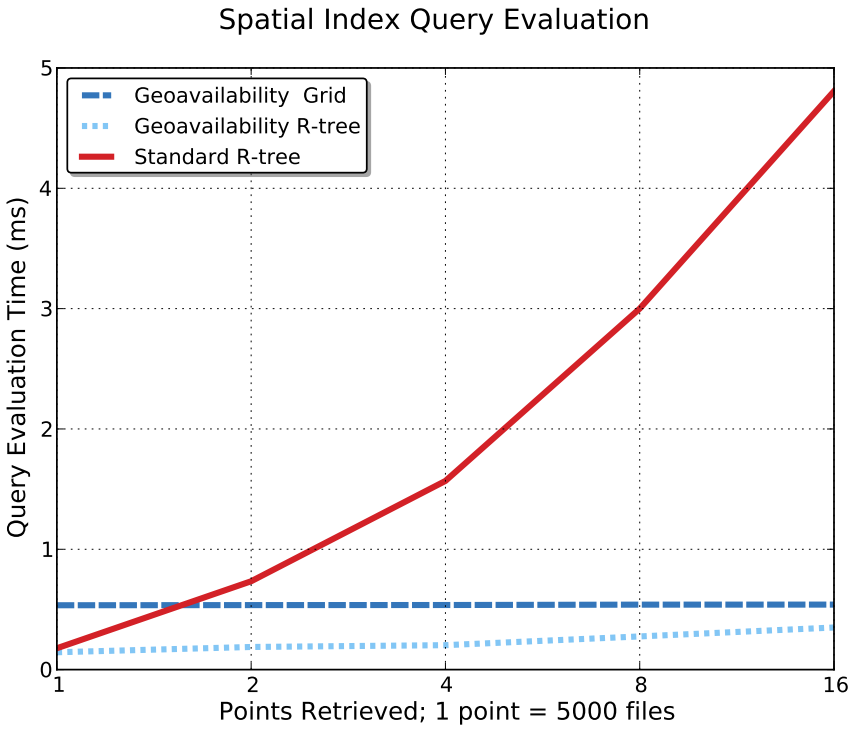


Fig. 2. Comparison of geoavailability grids, geoavailability R-trees, and a standard R-tree for retrieval operations. Results are averaged over 1000 runs.

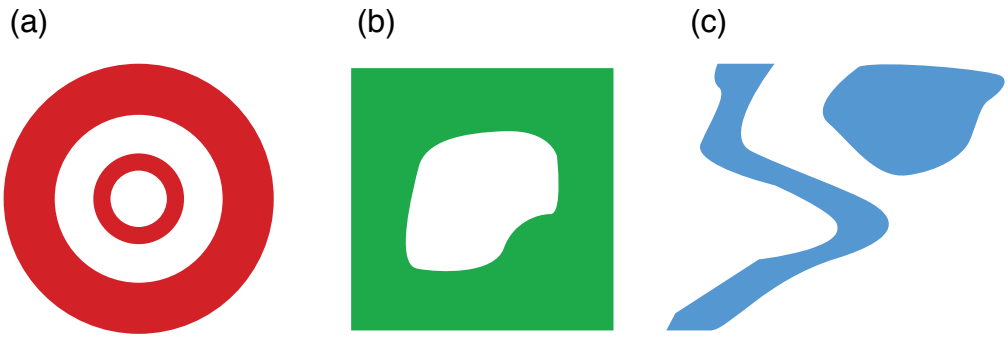


Fig. 3. Sample query geometry. From left to right: a radiation pattern (a), a geographical area surrounding a landmark (b), and a river/lake (c) using a combination of line segments and curves.

query geometry has been created and is ready to be evaluated by the system, it is flattened to a composite shape and serialized to a compact binary representation for transmission across the network. Figure 3 provides example shapes that can be used to retrieve information for a variety of use cases. When covering an area of approximately 50 km, shapes (a) and (b) can be transformed into a composite shape by the system in about 0.24ms, while the more complex geometry in shape (c) takes about 0.39ms to produce with CPU-only graphics routines (averaged over 1000 runs).

6.2 Support for Preset Query Geography and Shapefiles

For analyzing trends in climate conditions across the continental United States using the NOAA test dataset, we imported geometries for congressional districts, counties, and zip code tabulation areas (ZCTAs) from the Topologically Integrated Geographic Encoding and Referencing (TIGER) datasets provided by the United States Census Bureau. To support these datasets, we added support for industry-standard Esri shapefiles. Shapefiles can be quickly converted to geometry compatible with geoavailability grids and stored on storage nodes that manage relevant geographical regions. This functionality can also be used for business-specific geographic boundaries; for

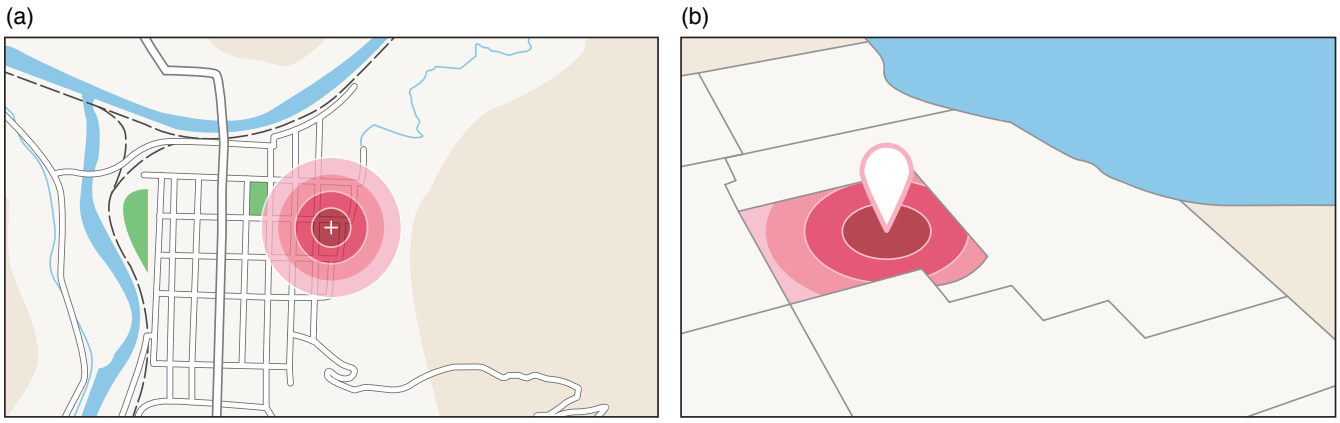


Fig. 4. (a) Proximity query starting from a given point on a street corner (indicated by a cross) and radiating outward with progressively wider-ranging query annuli; (b) query with boundaries constrained to the geometry of DuPage County, Illinois, USA.

instance, an electric utility company may wish to divide their service territory based on the coverage areas of their substations that provide power for consumers.

To analyze the performance of large-scale queries on preset geometric shapes, we used state boundaries imported from the TIGER datasets. Files containing humidity values above 70% were requested to constrain the queries and results were averaged over 1000 runs. Larger states required communicating with more storage nodes to resolve the queries: Texas, California, and Rhode Island contacted 18, 14, and 2 nodes, respectively, with complete metadata results streamed back to the client in 305.15ms, 265.07ms, and 24.81ms.

7 Proximity and Nearest Neighbor Searches

While the queries described thus far enable flexible retrieval of geospatial information, they do not account for situations where the desired geometry is not known at query time. This situation occurs frequently in geographic information systems (GIS) or cartographic visualizations; for instance, a user may wish to locate the nearest retail store that sells a particular range of products, but cannot estimate the breadth of his/her search. In these cases, the query geometry is unknown and the search is represented as a single starting coordinate pair. Additionally, proximity queries imply *priority* in their search results: the closest matching points should be returned first, followed by increasingly distant matches.

Given a pair of starting coordinates, proximity queries are evaluated by first checking the corresponding geoavailability grid location. If the bit in this location is set, then the search can be immediately evaluated by the storage node that contains information for the grid point. However, there are often situations where the coarse-grained geoavailability grid cell that was referenced does not contain matching information. In this case, circular query geometry is generated and centered over the starting coordinate pair. To broaden the scope of the search, progressively larger annuli (donut-shaped geometry) are evaluated against the geoavailability grids. This avoids re-querying areas that have already been inspected previously by the algorithm. Figure 4a illustrates the process behind evaluating a proximity query, with five iterations of the algorithm shown (including the initial grid cell check).

Each subsequent query annulus retrieves information that is streamed back to clients incrementally; the first results that a client receives will be the closest, followed by additional matches that are located geographically further from the original coordinate pair. With 25-bit grids, 8 iterations of the algorithm over a 50km region took 5.35ms with a standard deviation of 0.41ms when averaged over 1000 runs. Using the same parameters on a single-node installation of MongoDB 2.4 [9], the operation took 56.95ms with a standard deviation of 4.53ms. One key performance advantage of our proximity algorithm is that it does not require results to be collected and sorted as a post-processing step.

7.1 Geographically Constrained Proximity Queries

In circumstances that require a proximity search within a specific geographic region, we provide *geographically constrained* proximity queries. As with a standard proximity query, a geographically constrained search begins with an origin point and radiates outward to discover more information, but is limited to a particular set of geometric bounds. Much like a query against preset geography, geographically constrained proximity queries can take advantage of Esri shapefiles or datasets such as TIGER for defining spatial constraints. Figure 4b provides an

example of a spatially-constrained proximity query in DuPage County, Illinois. While there may be matching data points in the neighboring counties, they are excluded from the search.

7.2 Graphics Pipeline Optimizations

Our algorithm for servicing proximity queries makes extensive and frequent use of the underlying graphics framework, which can be accelerated by a compatible GPU. However, capable GPUs are still only occasionally found in most cloud deployments. Additionally, there are costs associated with transferring information in and out of graphics memory that must be considered when enabling GPU acceleration. When starting for the first time, storage nodes perform a hardware benchmark to determine the trade-off space for memory latency and image generation times, and use the results to autonomously choose the fastest rendering technique. In many cases, queries that span a small geographic region will be generated by the CPU, but operations on very large bitmaps will be offloaded to the GPU.

One factor that has a strong influence on bitmap generation times is the size of the geoavailability grids being used by the system. A larger geoavailability grid results in a proportionately larger "canvas" that will used for rendering query geometry, which consumes additional memory and processing time. In many cases, a query will span less than 1-5% of the entire geoavailability grid. To avoid the performance penalties associated with creating large images in memory, the maximum bounds of incoming query geometry are calculated and used to create a much smaller *query canvas* in memory that will be used for drawing operations. Once the canvas has been created, the query bitmap is generated and then shifted into place using geoavailability grid coordinates. This can provide significant performance gains; Table 3 compares the naive bitmap generation scheme with the optimized version for processing proximity queries with varying radii. Both algorithms were run on the CPU.

TABLE 3
Query bitmap generation times for proximity searches with varying radii, averaged over 1000 runs

Modifications	Creation Time (ms)	SD (ms)
Naive Algorithm, 25 km	30.63	7.34
Optimized, 50 km	0.85	0.26
Optimized, 25 km	0.30	0.12
Optimized, 5 km	0.09	0.05

8 Relevance Ranking

Results from proximity queries have an inherent relevance; points that are closer to the query centroid will generally be more relevant. However, there are several exceptions to this rule when dealing with a multidimensional dataset. For instance, a shopper searching for candles may prefer a candle store over a department store that sells a broader range of products. In a climate study, points with a temperature range of 308-310 Kelvin and a relative humidity of 50-60% may be more relevant for a particular investigation in a given subregion. These and other similar use cases led to the development of integrated *relevance ranking* in our query processing system.

Relevance rankings are provided by applying a *relevance function* on the results from a query. Galileo includes a selection of built-in relevance measures for spatial information, and also supports user-defined functions. For range queries, a *relevance gradient* can be provided to select highly relevant subsets of the data first, followed by the remaining information. This functionality, which effectively allows multi-tiered queries, has two primary advantages: rankings can be used by clients to sift or sort though information, and the most relevant data points are streamed back to clients first, obviating the need for manual sorting or pre-processing.

9 Related Work

MongoDB [9] is a distributed document store that also provides geospatial indexing capabilities. For load balancing and horizontal scalability, MongoDB supports sharding and dataset partitioning. Range queries, data replication, and MapReduce computations can also be handled natively by the system. While MongoDB and Galileo share several features, Galileo is generally intended to be used with spatiotemporal datasets containing multidimensional arrays, whereas MongoDB handles JSON-style documents and has some limitations or hard thresholds that can constrain extremely large collections of files.

SciDB [11] is a scalable scientific storage system that supports multidimensional data. SciDB focuses on incremental scalability for datasets containing petabytes of information. The system also provides built-in computation

and analysis tools, whereas Galileo only deals with storage operations; analysis is performed with an external distributed computation engine. Metadata is stored in a centralized *system catalog*, contrasting with the combination of feature graph and metadata graphs used in Galileo.

P2PR-Tree [12] is a P2P-based version of the R-Tree spatial index. The system is decentralized and can also service spatial queries while peers are leaving or joining the network. In P2PR-Tree, queries are routed to nodes that may have pertinent information, with a traversal through the network closely resembling a traversal through an R-Tree. This traversal pattern allows the system to cope with frequently added or removed nodes, but also involves additional routing steps that could increase latencies. Initially, the range of possible spatial values is broken up into *blocks*, with each block being statically divided into a pre-set number of groups. Nodes in the system are then divided into multiple levels of subgroups with neighboring peers maintaining more detailed information about one another. Each peer also maintains a local R-Tree for performing lookups on the data it holds. P2PR-Tree is well-suited for collections of information with geospatial properties, but does not support multidimensional datasets directly.

10 Conclusions

The geoavailability grid indexing scheme provides significant reductions in the search space of user-defined geometric queries in a distributed hash table. Instead of indexing every spatial location in the system, grid coordinates are converted to a coarse-grained compressed bitmap representation. Support for arbitrary geometric queries allows the system to account for natural, administrative, and political boundaries, as well as provide support for preset geographical queries and proximity-based searches. Our support for relevance ranking allows users to bias their queries along a particular dimension (e.g., proximity) or combination of dimensions. Overall, the approach detailed in this work provides an avenue for coping with extreme-scale data volumes across many distributed computing resources and allows fast and flexible retrieval of the information for analysis and processing.

Acknowledgments

This research has been supported by funding from the US Department of Homeland Security's Long Range program (HSHQDC-13-C-B0018) and the US National Science Foundation's Computer Systems Research Program (CNS-1253908).

References

[1] M. Malensek, S. Pallickara, and S. Pallickara, "Expressive query support for multidimensional data in distributed hash tables," in *Utility and Cloud Computing (UCC), 2012 Fifth IEEE International Conference on*, nov. 2012.
[2] ——, "Polygon-based query evaluation over geospatial data using distributed hash tables," in *Utility and Cloud Computing (UCC), 2013 Sixth IEEE International Conference on*, Dec. 2013.
[3] K. Ericson, S. Pallickara, and C. Anderson, "Analyzing electroencephalograms using cloud computing techniques," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 185–192.
[4] NOAA. (2013) The NAM. [Online]. Available: http://www.emc.ncep.noaa.gov/index.php?branch=NAM
[5] A. Guttman, *R-trees: A dynamic index structure for spatial searching*. ACM, 1984.
[6] Wikipedia Contributors. (2013) Geohash. [Online]. Available: http://en.wikipedia.org/wiki/Geohash
[7] D. Lemire *et al.*, "Sorting improves word-aligned bitmap indexes," *Data & Knowledge Engineering*, vol. 69, no. 1, pp. 3–28, 2010.
[8] K. Wu *et al.*, "Notes on design and implementation of compressed bit vectors," Lawrence Berkeley National Laboratory, Tech. Rep., 2001.
[9] mongoDB Developers, "Mongodb," *http://www.mongodb.org/*, 2013.
[10] JSI (java spatial index). [Online]. Available: http://jsi.sourceforge.net/
[11] P. Brown, "Overview of scidb: large scale array storage, processing and analysis," in *Proceedings of the 2010 international conference on Management of data*. ACM, 2010, pp. 963–968.
[12] A. Mondal *et al.*, "P2pr-tree: An r-tree-based spatial index for peer-to-peer environments," in *Current Trends in Database Technology-EDBT 2004 Workshops*. Springer, 2005.