Polygon-Based Query Evaluation over Geospatial Data Using Distributed Hash Tables

Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara
Department of Computer Science
Colorado State University
Fort Collins, USA
{malensek, sangmi, shrideep}@cs.colostate.edu

***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 polygonal bounds on a distributed hash table architecture. Our solution involves novel distribution and partitioning of these voluminous datasets, thus enabling 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. This results in improved response times and more effective utilization of available resources. Geoavailability grids are also applicable in non-distributed settings for local lookup functionality, performing competitively with other leading spatial indexing technology.

***Index Terms*—Distributed Hash Tables, Cloud Infrastructure, Polygonal Queries, Time Series Data**

I. 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 mechanisms 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. These properties have led to DHTs underpinning a wide range of cloud-scale infrastructures.

During analysis, there is often a need for providing spatial bounds of interest in the form of user-specified polygon shapes. Such polygonal queries can correspond to administrative or natural boundaries, and provide greater freedom to apply various forms 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 polygon-based query support in question must be decoupled from the generation and storage of data.

A. Research Challenges

We consider the problem of fast and scalable evaluation of arbitrarily-shaped polygonal queries over time series datasets with geospatial properties. The challenges involved in doing so include:

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.
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.

B. 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 synchronize and update global views of available data? The resolution of the data structures has a direct correlation with update traffic.
3) How can we support polygon-based queries without compromising the key strengths of DHTs?
4) 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?

C. Overview of Approach

The approach described in this paper is based on our hierarchical DHT implementation, Galileo [1], [2], [3]. Galileo is designed for high-throughput management of multidimensional data streams. The network hierarchy in Galileo allows for novel data partitioning configurations, a property that is exploited in this work to help reduce the search space of distributed queries by decreasing the number of machines that must be contacted during query evaluation. Data dispersion is based on the Geohash [4] geocoding scheme, which creates 1-dimensional string representations of 2-dimensional geographic bounding boxes. The number of characters in a Geohash corresponds to its geographical scope and precision, with longer Geohash strings representing finer-grained spatial regions.

To create an overall view of the spatial locations of data stored in the system, we provide a globally-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.

Since a number of geoavailability grids are used to track locations of data for the entire system, their memory consumption and traversal times could be prohibitively expensive. To alleviate this issue, we convert the grid to a bitmap representation where a single bit signifies the presence of data within a given spatial region. Bitmaps are highly amenable to specialized compression, which further reduces the size of the data structures while also increasing the speed of bitwise operations.

There is an inherent trade-off in the granularity of the geoavailability grid and the efficiency of storage and query evaluations. A coarser grid will reduce memory consumption and accelerate query evaluation, but may not provide a significant reduction in search space. The factors that influence these variables are investigated thoroughly in this work.

D. Paper Contributions

This paper demonstrates the effectiveness of using globally-distributed, bitmap-based indexes to evaluate polygonal queries, along with how the distribution of data can alleviate indexing load. The solution described is scalable and can cope with increases in both data volumes and the number of nodes that comprise the storage network. By eliminating nodes from the search space that do not contain data that will satisfy a given query, unnecessary communication and processing overheads can be avoided. This functionality is tested empirically on a dataset consisting of one billion files.

Additionally, our framework accelerates queries by evaluating them concurrently across a number of relevant nodes while supporting expressive range-based and exact-match retrievals on feature values in addition to polygon boundaries. When available, the system can harness GPU acceleration for bitmap generation on the host machines. Bitmap compression and GPU acceleration are configured autonomously by the system at runtime based on availability and storage conditions. Most importantly, data generation and query specification are completely decoupled in our solution. This makes our approach applicable to other DHT-based networks or storage frameworks, as well as non-distributed applications; depending on storage constraints, our index can also outperform the established R-tree spatial index in a local (single node) environment.

E. Paper Organization

The remainder of the paper is organized as follows. Section II introduces our DHT implementation and its data structures. Section III details our bitmap-based index, the geoavailability grid. Section IV explores using the geoavailability grid for reducing the search space of distributed queries, followed by Section V that explores local retrieval operations by contrasting with another spatial index. Section VI surveys related work, followed by Section VII with discussion of our conclusions and future work.

II. System Overview

Galileo is a distributed storage framework that is a modeled as a hierarchical distributed hash table (DHT). Contrasting with the standard DHT design seen in Chord [5] or Pastry [6] where a hash space is partitioned among a number of computing resources, or *nodes*, Galileo employs multiple hash functions to subdivide and create logical groupings of resources. Additionally, Galileo is a *zero-hop* DHT, meaning that requests are not forwarded through intermediate nodes in the network and instead are routed directly to their destination, a feature also seen in DHTs such as Apache Cassandra [7] and Amazon Dynamo [8]. This hybrid design allows for a scale-out architecture that supports functionality generally not implemented in traditional DHT-based systems.

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 a number of additional *features* of interest. Galileo supports a range of scientific storage formats such as NetCDF [9] or HDF5 [10] alongside its own native format. For feature value retrievals, the system allows both exact-match and range-based queries through a layered indexing strategy

that incorporates a global *feature graph* and local *metadata graph* instances.

A. Partitioning

Galileo supports flexible, user-configurable data partitioning schemes. Unlike a standard DHT, the hash functions used for partitioning are not required to support retrieval operations as well; instead, queries can be resolved using the system's multilayer index. This approach allows for novel load distribution configurations and hierarchical network layouts.

Individual *storage nodes* in Galileo can be placed into a number of *groups* or *subgroups* in the network. Depending on the datasets being stored, a hierarchy of disparate hash functions can be used for data placement. In this study, each group is assigned a portion of the overall geography being managed by the system, while node placement within groups is determined by a SHA-1 hash of feature values. This allows for a reasonably balanced distribution of load while also reducing the search space of geospatial queries with known coordinates.

B. 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. Results from the graph are returned in the form of subgraphs called *datasets*, which can be traversed, modified, and used to retrieve files from the system. The metadata graph is composed of numeric or string-based values and allows for quick evaluation of both exact-match and range queries.

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 be evaluated against local metadata graphs. In most cases, this optimization provides a dramatic reduction in the number of nodes that must be contacted to evaluate a given query operation.

C. Experimental Configuration

In this study, we used real-world data from the National Oceanic and Atmospheric Administration (NOAA) North American Mesoscale Forecast System (NAM) [11] project. Using our NetCDF input plugin, we sampled from this dataset to create a test dataset consisting 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. Galileo was run under the OpenJDK Java runtime version 1.7.0_25.

III. Indexing: The Geoavailability Grid

Indexing in a distributed environment with highly voluminous datasets poses a number of challenges; 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 [12] is commonly employed in non-distributed applications for spatial indexing due to its speed and efficiency, but has a number of 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 a number of 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. They consist of a vector of bits (represented by the set $\{0, 1\}$) for a given spatial area. 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 relational database systems, decision support systems [13], and data warehousing [14].

A. Geocoding

To partition information in our DHT and provide a coarse-grained representation of its spatial properties, we use the Geohash [4] 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, the decimal coordinates of 44.509° N, -110.331° W would map to the Geohash string *9xct1qe7*, 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. Quad Trees [15] or the OpenPostcode [16] algorithm could be used to achieve similar results.

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 individual nodes can exclude information from their geoavailability grids that lies outside their geographic scope.

B. 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 on a per-node basis, enabling queries to distinguish between different feature types. For situations where there is a reading present for every type, a catch-all geoavailability grid is generated autonomously to reduce the overall number of bits set in each grid.

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:

$$9xct1q = \underbrace{01001\ 11101}_{\text{Group Hash}}\ \underbrace{01011\ 11001\ 00001\ 10110}_{\text{Location in Bitmap Index}}$$

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 an example geographic region could be represented as a geoavailability grid.

C. Compression

Each node in the system has a number of associated index bitmaps for the geographic region under its purview. The bitmaps are distributed across all the nodes in the cluster to ensure that (1) every node in the system is capable of servicing queries, and (2) a node failure does not affect lookup capabilities. This means that the memory consumed by the bitmaps must be low; continually exchanging large amounts of information between nodes is inefficient and reduces the speed of index propagation.

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 CONCISE [17] or WAH [18]. In this work, we use the Enhanced Word-Aligned Hybrid (EWAH) compression scheme as described by Lemire et al. [19] to reduce the effective sizes of our bitmaps. EWAH was chosen due to the compression ratios it achieved on our dataset and its relative speed. Table I 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.

TABLE I
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

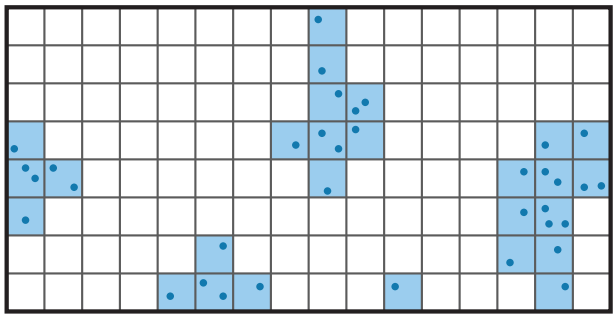
Compressed bitmaps are somewhat unique in that they generally do not require decompression before processing occurs. In fact, compression can often speed up bitwise operations. Due to the differences in performance observed across the available bitmap compression algorithms, we provide a universal interface that allows the underlying bitmap representation of a geoavailability grid to be changed at runtime or during initial system configuration.

D. 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. The update interval and maximum number of unpublished updates are configurable parameters depending on network capacity and desired consistency characteristics.

Creating an update involves performing an XOR ($\oplus$) operation on the current index i_c and the previous index state i_p, which produces a set of update bits u that will be relayed to peer nodes: $i_c \oplus i_p = u$. When an update is received at a peer, it can be applied by simply performing an XOR between the update and the current copy of the remote index ($u \oplus i_p = i_c$). The latest index version number and a checksum are also included in an update message to help ensure consistency. Updates are compressed in the same manner as the indexes, meaning that they generally consume a minimal amount of space; Table II contains update sizes (including the version number and checksum) for different amounts of new data points that were added in random spatial locations over a 20-bit Geohash region. A single update will always have a size of 36 bytes since only one bit is set in the compressed bitmap.

Random updates represent an approximate worst case in terms of message size because the likelihood of an existing bitmap location already being occupied is low; after all, new



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 a number of data points, with its geoavailability grid (right).

TABLE II
INDEX UPDATE SIZES, AVERAGED OVER 1000 RUNS

Modifications	Update Size (bytes)	SD (bytes)
1	36.0	0.0
10	179.9	1.0
100	1609.3	10.8
1000	15089.0	98.4

files that fall within a spatial range already covered by the geoavailability grid do not require any updates to be transmitted. With our test dataset, storage nodes were responsible for approximately 5000 unique bitmap locations on average over the lifetime of the cluster, which would only require about five 1000-bit updates.

One contributing factor in the efficiency of bitmap representations is the bit-level parallelism that can be exploited while evaluating bitwise operations. This property makes creating and applying updates extremely fast; considering the update sizes and their processing times, keeping the geoavailability grid up-to-date is mostly a function of network latencies. To demonstrate this, we created and applied updates of size 1, 10, 100, and 1000 and averaged the results over 1000 trials. No matter the size of the update, generation took *0.32 ms* on average with a standard deviation of 0.03, and applying the updates took *0.19 ms* with a standard deviation of 0.05.

IV. Retrieval: Polygon-Based Query Evaluation

Once spatial data has been indexed in geoavailability grids at each storage node, the system can evaluate user-defined geospatial queries derived from polygons surrounding areas of interest. Queries can also contain ranges or exact values of various features to help further reduce the search space. Geospatial query evaluation in Galileo proceeds as follows:

1) A user submits a polygon to retrieve data from.
2) The query is *decomposed* into a number of subqueries by intersecting it with the geometry each group of nodes is responsible for.
3) The geoavailability grids are consulted to determine if data may be available, eliminating any irrelevant nodes.
4) Specified feature values or ranges are queried from the feature graph, allowing additional reduction of the search space.
5) Subqueries are submitted to the remaining set of relevant nodes for evaluation.

A. 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. Figure 2 illustrates this procedure.

Decomposing queries in this manner has multiple 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 a number of groups. After decomposing the query, geoavailability grids can be consulted to further reduce the search space.

B. 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

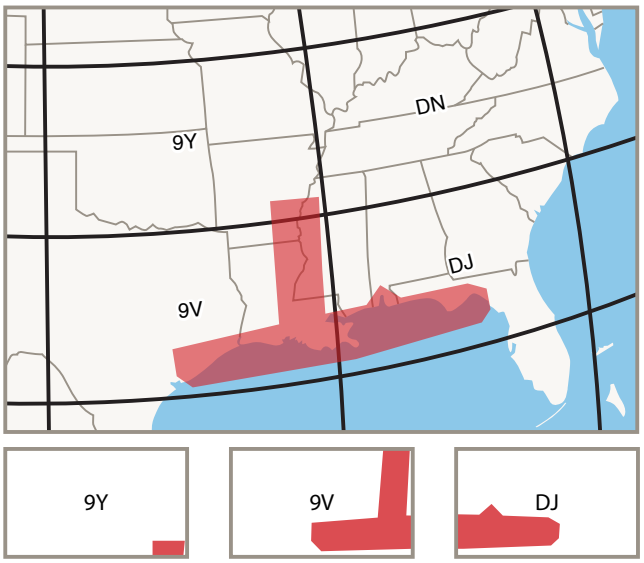


Fig. 2. A query following the Mississippi river through the Louisiana area and along the shoreline in the Gulf of Mexico (shown in red) decomposed into three subqueries across Geohash group boundaries.

available. Another benefit of this strategy is that queries can easily be visualized. Table III contains information on how long it takes to generate a query bitmap based on index resolution, which is the primary factor involved when drawing query geometry at scale. For this benchmark, the subquery polygon shown in Figure 2 for Geohash area *9V* was converted into a query bitmap.

TABLE III
QUERY BITMAP CREATION TIMES, AVERAGED OVER 1000 RUNS

Resolution	Bitmap Generation (ms)	SD (ms)
15-bit	0.45	0.36
20-bit	1.81	1.14
25-bit	56.31	4.78

While the exact precision of a Geohash is largely dependent on its particular location on the globe, a precision of 25 bits equates to a region of about 230 by 150 meters. For coarser-grained query bitmaps, the storage and processing times decrease. Figure 3 illustrates the benefits and drawbacks associated with increasing or decreasing query bitmap resolution; while false positives are less likely with a higher resolution, the memory consumed by the geoavailability grids also increases.

Once a query bitmap has been obtained, evaluating the presence of relevant data within the polygon boundaries is extremely simple; a logical AND ($\wedge$) is performed between geoavailability grids (GG) and the query bitmap (QB). 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. In other words, if $GG \wedge QB = \{\}$ for a particular storage node, then it can safely be eliminated from the search without requiring any communication. Table IV contains timing information for processing a geoavailability lookup, which involves several AND operations.

TABLE IV
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

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 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.

V. LOCAL RETRIEVAL: R-TREE EVALUATION

Once a query has been decomposed and evaluated, it can be processed in parallel as a set of subqueries at relevant storage nodes. This process is facilitated by the metadata graph for range-based and exact-match queries of feature values, but additional indexing infrastructure is required to provide results for polygon-based spatial queries. The R-tree [12] is a fast and efficient spatial index that has seen considerable usage and development in the geospatial domain. Similar to B-trees, R-trees are a balanced search tree and organize data into pages. The basic unit of storage is a rectangle, so polygon-based shapes or queries are first converted to their minimum bounding rectangle (MBR) before use with the index.

R-trees provide an appealing alternative to using geoavailability grids for local retrieval because their accuracy is not based on a preconfigured resolution, and the average complexity of a lookup with maximum page size M is $O(log_M n)$ versus $O(n)$ for a bitmap representation. However, results returned from a geoavailability evaluation do not require post-processing to eliminate rectangles or points that were present in the query MBR but do not fall within the query polygon. Additionally, query bitmaps have already been generated for the geoavailability grids in previous indexing steps, so a local lookup only involves processing a number of AND operations.

To assess the performance of these two indexing techniques for local retrieval, we generated a multi-MBR approximation of the polygon subquery on region *9V* shown in Figure 2 for evaluation on an R-tree, and submitted its previously-computed query bitmap to our 25-bit geoavailability grids. We used the Java Spatial Index (JSI) [20] implementation of R-tree in these benchmarks due to its focus on performance. To test the indexing methods under different retrieval loads,

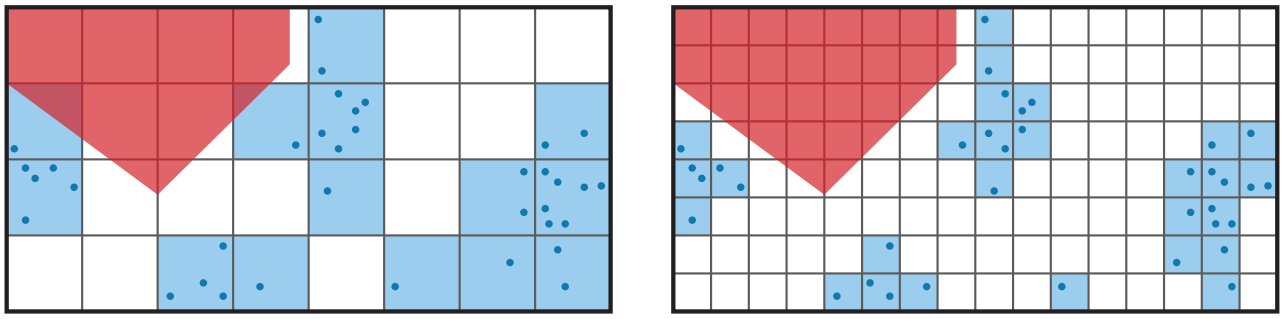


Fig. 3. Two different query bitmap granularities and an example query polygon. Finer-grained bitmaps generally increase the amount of search space reduction that can be achieved at the cost of consuming more memory.

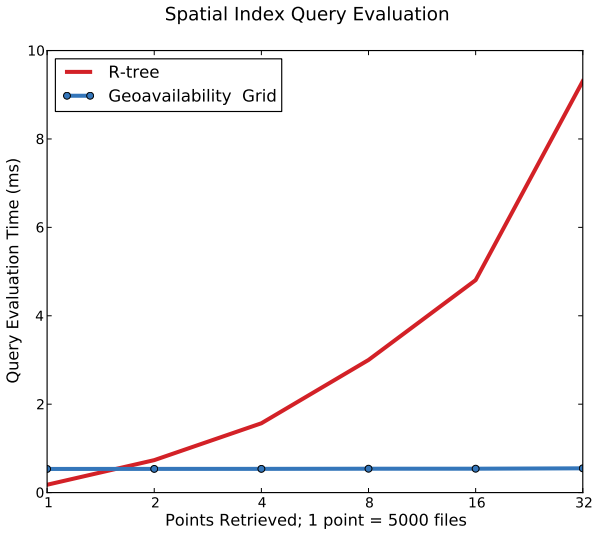


Fig. 4. Comparison of geoavailability grids and R-tree for retrieval operations. Results are averaged over 1000 runs.

we scaled the query polygon to include increasingly larger amounts of points across all feature values and averaged the results. Figure 4 illustrates the performance of both strategies when retrieving points from our test dataset, with each point containing about 5000 files. The figure provides a number of insights into the performance profiles of both strategies: geoavailability grids tend to have very predictable performance, and can outperform R-trees when retrieving large amounts of files. On the other hand, the R-tree performs well with a small number of files, indicating it may be useful for more sparsely-populated datasets.

One of the strengths of our geoavailability grid implementation is bounded memory consumption: for a given resolution, there is a maximum size of the resulting bitmap whether compression is used or not. We tested the memory consumption of three different geoavailability grid resolutions and an R-tree to gain insights on their resource consumption. Since our billion-file NOAA dataset is divided among 48 storage nodes, approximately 20 million points were indexed and tested at each node. Table V contains the results of this benchmark.

TABLE V
MEMORY CONSUMPTION OF GEOAVAILABILITY GRIDS AND A JSI R-TREE.

Implementation	Memory Consumed (KB)
15-bit Geoavailability grid	3.98
20-bit Geoavailability grid	44.37
25-bit Geoavailability grid	56.03
JSI R-tree	1120634.88

Geoavailability grids are clearly more compact than the R-tree used in this benchmark, making them a good candidate for virtualized infrastructure or other scenarios where memory is scarce. On the other hand, an R-tree does not store its contents in a reduced-resolution form, making it better-suited for situations that require extreme query precision. Ultimately, choosing between an R-tree or geoavailability grid is highly dependent on the use case scenarios being dealt with and the constraints of the operating environment; for this reason, we created a spatial index interface in Galileo that allows the local retrieval index to be changed depending on the use case.

VI. Related Work

Cassandra [7] is similar to Galileo in its network layout and storage capabilities. It allows users to create their own partitioning schemes, and has also leveraged Geohashes to provide spatial queries and indexing. Contrasting with Galileo, the partitioning algorithm used in Cassandra directly affects possible retrieval operations; using the random data partitioner backed by a simple hash algorithm does not allow for range queries or later reconfiguration of the partitioning scheme. While Cassandra supports tabular, multidimensional data representations, it is primarily designed for text rather than feature vectors or device readings.

MongoDB [21] is a distributed document store that includes rich geospatial indexing capabilities. Its storage format supports dynamic schemas with a binary representation similar to JSON. Like Galileo, MongoDB utilizes Geohashes for its spatial index. The Geohashes can then be stored directly in

MongoDB's B-tree index for lookup operations. For load balancing and scalability, MongoDB supports sharding that divides ranges of data between nodes in the system. Range queries, data replication, and MapReduce are also supported.

P2PR-Tree [22] provides 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. 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.

SD-Rtree [23] aims to provide a scalable distributed R-Tree implementation. The resulting system takes inspiration from both R-Trees and AVL trees, and provides a number of access methods with differing scalability and performance. Similar to a traditional R-Tree, the SD-Rtree inserts information at its leaf nodes, splitting them when they overflow. This property greatly increases the amount of communication required in the early stages of the tree creation, but gradually decreases as more information is stored and the tree structure settles. A key strategy incorporated in SD-Rtree is client-side caching to speed up queries and reduce the amount of traffic directed towards the root of the tree; this eventually-consistent approach results in a large reduction of overall messages sent during insertions and query resolution.

VII. Conclusions and Future Work

A. Conclusions

When combined with hierarchical data partitioning, our geoavailability grid indexing scheme provides significant reductions in the search space of user-defined polygonal queries in a distributed hash table. Instead of indexing every spatial location in the system, grid coordinates are converted to a coarser-grained compressed bitmap representation. The low memory footprint of the data structures used in this strategy makes distributing snapshots of global information fast and efficient, even for large datasets. This allows any node in the DHT to facilitate distributed queries by first eliminating any nodes that do not contain relevant data from the search before submitting subqueries for local retrieval. At each storage node, local retrievals can also be evaluated using the geoavailability grid, an approach that compares favorably with the established R-tree spatial index depending on use case and storage properties. Overall, our approach provides an avenue for coping with extreme-scale data volumes across a number of distributed computing resources and allows fast and flexible retrieval of the information for analysis and processing.

B. Future Work

One key insight derived from this work is that low-resolution geoavailability grids are extremely fast and space-efficient. This property could be exploited by allowing variable-resolution subgrids to be embedded within the geoavailability grid for regions that require higher precision. This would enable our indexing scheme to achieve even higher accuracy while maintaining its positive features for most use cases. Furthermore, the variable precision subgrids are an appealing target for enabling autonomous configuration by the system at run time, allowing for dynamic improvements in both search space reduction and query response times without user intervention.

References

[1] M. Malensek, S. Pallickara, and S. Pallickara, "Exploiting geospatial and chronological characteristics in data streams to enable efficient storage and retrievals," *Future Generation Computer Systems*, 2012.

[2] ——, "Expressive query support for multidimensional data in distributed hash tables," in *Utility and Cloud Computing (UCC), 2012 Fifth IEEE International Conference on*, nov. 2012.

[3] ——, "Autonomously improving query evaluations over multidimensional data in distributed hash tables," in *Proceedings of the 2013 ACM International Conference on Cloud and Autonomic Computing*, aug. 2013, (To appear).

[4] Wikipedia Contributors. (2013) Geohash. [Online]. Available: http://en.wikipedia.org/wiki/Geohash

[5] I. Stoica *et al.*, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[6] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*. Springer, 2001, pp. 329–350.

[7] A. Lakshman *et al.*, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Op. Sys. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.

[8] D. Hastorun *et al.*, "Dynamo: amazon's highly available key-value store," in *SOSP*. Citeseer, 2007.

[9] R. Rew *et al.*, "Netcdf: an interface for scientific data access," *Computer Graphics and Applications, IEEE*, vol. 10, no. 4, pp. 76–82, 1990.

[10] Q. Koziol *et al.*, "Hdf5–a new generation of hdf: Reference manual and user guide," *National Center for Supercomputing Applications, Champaign, Illinois, USA, http://hdf. ncsa. uiuc. edu/nra/HDF5*, 1998.

[11] National Oceanic and Atmospheric Administration. (2013) The north american mesoscale forecast system. [Online]. Available: http://www.emc.ncep.noaa.gov/index.php?branch=NAM

[12] A. Guttman, *R-trees: A dynamic index structure for spatial searching*. ACM, 1984, vol. 14, no. 2.

[13] C.-Y. Chan and Y. E. Ioannidis, "Bitmap index design and evaluation," *ACM SIGMOD Record*, vol. 27, no. 2, pp. 355–366, 1998.

[14] T. L. Lopes Siqueira, R. R. Ciferri *et al.*, "A spatial bitmap-based index for geographical data warehouses," in *Proceedings of the 2009 ACM symposium on Applied Computing*. ACM, 2009, pp. 1336–1342.

[15] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[16] Open Postcode Ireland. (2013) Ireland's postcode. [Online]. Available: http://www.openpostcode.org/

[17] A. Colantonio and R. Di Pietro, "Concise: Compressed 'n' composable integer set," *Info. Proc. Ltrs*, vol. 110, no. 16, pp. 644–650, 2010.

[18] K. Wu, E. J. Otoo, and A. Shoshani, "An efficient compression scheme for bitmap indices," 2004.

[19] D. Lemire *et al.*, "Sorting improves word-aligned bitmap indexes," *Data & Knowledge Engineering*, vol. 69, no. 1, pp. 3–28, 2010.

[20] Jsi (java spatial index) rtree library. [Online]. Available: http://jsi.sourceforge.net/

[21] mongoDB Developers, "Mongodb," *http://www.mongodb.org/*, 2013.

[22] 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, pp. 516–525.

[23] C. du Mouza, W. Litwin, and P. Rigaux, "Sd-rtree: A scalable distributed rtree," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 296–305.