GeoLens: Enabling Interactive Visual Analytics over Large-scale, Multidimensional Geospatial Datasets

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

***Abstract*— With the rapid increase of scientific data volumes, interactive tools that enable effective visual representation for scientists are needed. This is critical when scientists are manipulating voluminous datasets and especially when they need to explore datasets interactively to develop their hypotheses. In this paper, we present an interactive visual analytics framework, GeoLens. GeoLens provides fast and expressive interactions with voluminous geo-spatial datasets. We provide an expressive visual query evaluation scheme to support advanced interactive visual analytics technique, such as brushing and linking. To achieve this, we designed and developed the Geohash based image tile generation algorithm that automatically adjusts the range of data to access based on the minimum acceptable size of the image tile. In addition, we have also designed an autonomous histogram generation algorithm that generates histograms of user-defined data subsets that do not have pre-computed data properties. Using our approach, applications can generate histograms of datasets containing millions of data points with sub-second latency. The work builds on our visual query coordinating scheme that evaluates geo-spatial query and orchestrates data aggregation in a distributed storage environment while preserving data locality and minimizing data movements. This paper includes empirical benchmarks of our framework encompassing a billion-file dataset published by the National Climactic Data Center.**

Keywords- [Visual analytics, distributed hash tables, Geospatial datasets, interactive analysis]

I. INTRODUCTION

Higher resolution displays are useful to understand and analyze complicated natural phenomena. However, as the scale of the dataset reaches human perceptual or cognitive limits, traditional data visualization tools are not efficient enough for the task of providing an intuitive and interactive sketch of the dataset. This is especially true for scientific datasets often used in simulations and analyses. Here, interactive explorations are a critical tool to develop scientific hypothesis, explore validity of models, and to reduce the time between further experiments.

Many datasets are too large to fit in memory or to even be stored on a single machine. Current data centers comprise large numbers of machines and the data is dispersed over them. At this scale, interactive visual analytics is often limited by the data access capability. Interactive visual analytics encompasses efficient query evaluation, effective data transfer, and distributed data processing for implementing visualization algorithms.

In this paper, we address the problem of scalable and interactive explorations of voluminous, distributed datasets

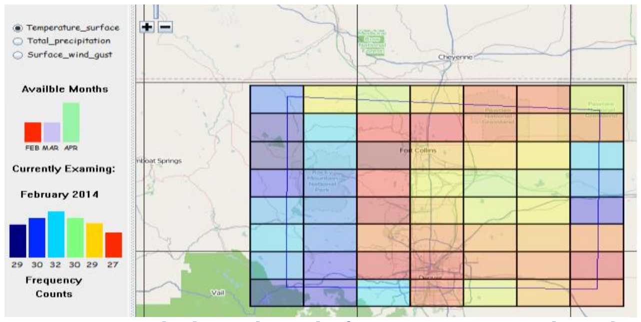

Figure 1. GeoLens displaying the result of a query encompassing the north-eastern part of Colorado. Here we see available months and features, as well as, the current geohash tiles and histogram counts for the month of February in 2014.

using an integrated approach that encompasses visualization algorithms and efficient evaluation of queries. Our approach also enables interactive visual analytics techniques such as brushing and linking over distributed raw datasets. Brushing and linking refers to the visualization technique that combines different visualization methods to overcome the shortcomings of using a single approach exclusively and in isolation. Users can select a subset of the data item using the brushing technique, and the user's brushing actions are connected via linking. [1, 2] To support this feature, the query should be expressive enough to represent the relations between the visual components, and allow for immediate traversal between multiple views. The system should also cope with the perceptual limit of the data visualization autonomously. The visual acuity with high-resolution display has imposed a perceptual scalability limit [3]. If the resolution of image is so high that an individual pixel cannot be seen, an even higher resolution is unlikely to be beneficial, regardless of how close a user is to a display.

We have explored the brushing and linking over heat maps and histograms of features. As depicted in Figure 1, here the targeted dataset is specified as the result of query generated by users. Since the targeted dataset is customizable, several dataset properties, such as the max/min or frequency counts, are not available until the query result is ready. Likewise, the scale of dataset has a wide range: from few kilobytes to multiple terabytes. In many cases, the pixel representation is much more compact than the raw data formats.

GeoLens is a framework for the scalable interactive visual analytics. To address perceptual scalability over geospatial datasets, we have developed algorithms for geohash based self-adjustable data tiles and autonomous histogram generation. Geohash based self-adjustable data tiles enable the system to access the required dataset based on the effective resolution of the display automatically. With autonomous histogram generation, the histogram of a feature will be generated without having any pre-computed data properties. We have included comparisons of several well-known histogram creation algorithms and how effective and scalable each algorithm is when they are applied in a distributed storage setting. Our results show that GeoLens provides sub-second latency to generate histogram for millions of data points with a reasonable accuracy.

To address the scalability for evaluating this advanced query in a distributed file system, we have developed a visual query coordinator. The visual query coordinator generates query results that are translated into visual components on the server-side. The visual query coordinator is selected from the set of available storage nodes and it is responsible for aggregating values, tracking visualization algorithms, and ensuring load balancing within the storage cluster.

We evaluated our approach in the context of our Galileo system [4-7]. Galileo is a scalable storage framework for managing multidimensional geospatial time-series data for scientific applications. Galileo stores blocks of multi-dimensional arrays with temporal and geospatial metadata alongside the attributes. Users are allowed to query these datasets by specifying multi-dimensional queries that specify bounds or wildcards for one or more dimensions corresponding to the observations/features stored within the system. Queries supported by Galileo include exact match queries, range queries, and approximate queries.

A. *Scientific Challenges*

This paper addresses the problem of interactive visual analytics over voluminous geospatial datasets. The challenges involved in accomplishing this include the following:

- The datasets are voluminous and must be dispersed over a number of distributed storage nodes.
- The use of traditional data visualizations over such voluminous datasets can incur high latency.
- Existing visualization applications are limited in their ability to facilitate perceptual understanding of datasets at this scale.
- Dataset properties such as distribution of features are unknown and may continuously evolve.
- For the integrated visual analytics technique such as brushing and linking, the relation between visual components is complicated and must be customizable based on the design of the application.

B. *Research Questions*

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

- How can we ensure fast visual query evaluations over distributed datasets so that we can support interactive visual analytics?
- How can we adjust the resolution of the image tile within the user's perceptual limit?
- How can we provide flexible and expressive system data structures so that they are amenable for interactive visual analytics techniques?
- How can we avoid large amounts of data transfers during visualization and processing? Given the data volumes involved, the transfer costs can introduce considerable delays.
- How can we maximize data locality during visual query evaluations?
- How can we effectively balance query workloads so that a large number of visual query evaluations can be performed concurrently?

C. *Overview of Approach*

The approach described in this paper is based on our distributed storage framework, Galileo [4]. Galileo is a hierarchical distributed hash table (DHT) that provides support for high-throughput management of voluminous, multidimensional observational data streams. Datasets managed by Galileo are partitioned and dispersed over a large cluster of commodity machines using the geohash encoding scheme to preserve geospatial proximity of data points. Query evaluations are assisted by memory-resident metadata graphs at each storage node.

To support perceptual scalability for the geospatial dataset, GeoLens provides geohash based self-adjustable data tiles. Based on the user's display and the zooming actions, the size of the image tile is translated to the length of geohash code. In the geohash algorithm, the length of the code represents the resolution of the geospatial region: a longer code has higher resolution, and correspondingly a smaller geospatial grid. No matter what the resolution the client selects, the final volume of data delivered to the user will be approximately the same in GeoLens. This feature is directly related to data locality, because Galileo disperses and stores the dataset based on the geohash values associated with the data points. In general, the resolution of geohash used for the data dispersion tends to be lower than the one used for the image tile. Generating a single image tile requires minimal data movements between the storage nodes.

Our autonomous histogram generation is a distributed algorithm that generates histograms from the dataset without any pre-computed or preserved data properties. A histogram represents frequency distribution using rectangles whose widths represent class intervals and whose areas are proportional to the corresponding frequencies [8]. Providing meaningful widths based on the complete data distribution is critical to achieving an effective histogram display. Since the targeted data is available only after the initial query result is ready, the algorithm cannot be performed offline. We have considered three types of algorithms: the N-Squared rule [9], the Freedman–Diaconis' choice (Freedman) [10], and Histogram Bin-width Optimization [11] to find an algorithm that scales well with acceptable latency and accuracy. GeoLens utilizes the Freedman algorithm and to provide sub-second latency for millions of data points.

In GeoLens, we define a visual query as a query that generates results that are processed by interactive data visual analytics tools. This involves data query evaluations, data aggregation, and transforming results to interoperate with the visual components. The query also involves distributed data processing such as autonomous histogram generation. To perform visual query evaluations at scale, we have devised a visual query coordinator. As soon as the user's visual query arrives at Galileo, one of the nodes (usually, the node containing the largest amount of data points) will act as a proxy for the client to collect the query results that are image tiles being generated by the other nodes. This architecture allows for dispersion of processing loads since the data is processed at the nodes that hold the data. Data movements between the storage nodes are reduced because of data locality during processing. The server-side processing also reduces the amount of data that needs to be transferred between client and server while minimizing the amount of data processing in the client application.

D. Paper Contributions

This paper presents the design of the GeoLens visual analytics framework for voluminous, multivariate, geospatial datasets. The paper explores the use of brushing and linking technique with heat maps and histograms of the time series features using GeoLens. Our approach addresses the scalability in terms of the visual perception and distributed processing. Our distributed algorithm can generate histograms of dispersed datasets without pre-computed data properties. Our Geohash based self-adjustable image tiles algorithm addresses visual perceptive scalability by keeping the size of the final results minimum and constant. Our visual query coordinator approach preserves data locality while minimizing data movements; this allows for faster visualization. Our evaluations target latency for performing visual queries at various scales of query results to demonstrate its feasibility in various settings.

E. Paper Organization

The remainder of this paper is organized as follows. In section II, we provide a background of this research. In section III, we describe the query evaluation scheme in GeoLens. Section IV discusses the brushing and linking feature over heat maps and histograms. We explain each of the aggregation methods, and how VisGraph of GeoLens links these methods. Section V provides related works. A performance evaluation of various aspects of the system is presented in Section VI. Finally, conclusions and future work are outlined in Section VII.

II. BACKGROUND

A. Galileo

For our storage requirements, we utilize Galileo, a distributed data storage framework for voluminous multi-dimensional, geospatial, time-series datasets. Data can be streamed to Galileo from any sort of device designed to observe facets of our environment, such as sensors or satellites, and Galileo will handle dispersion and indexing.

1) Network Topology

Galileo is modeled as a distributed hash table (DHT). However, unlike standard DHTs such as Tapestry [12] or Chord [13], Galileo employs a two tier hashing scheme for block distribution among multiple machines, or nodes. Galileo is a zero-hop DHT, a feature seen in other implementations such as Amazon's Dynamo [14], meaning requests are sent directly to their destination, rather than visiting intermediate nodes.

Galileo is well suited for storing and processing voluminous, multivariate datasets containing spatial and temporal information, and additional features of interest. Galileo can support a variety of common data formats, such as NetCDF [15], and can handle a variety of query types, including both exact-match and range-based queries. However, unlike other DHTs, Galileo does not use a generic hashing scheme for data distribution, and instead exploits geospatial characteristics in the data for distribution.

2) Geohashing

By employing a generic hashing scheme, we can achieve the desired result of distributing the data evenly across our nodes. However, we can preserve the geospatial information in the data, and achieve data dispersion, while grouping similar points, if we employ a geohash algorithm [16] as our first-tier hashing scheme. The purpose of the geohash algorithm is to divide the earth into an arbitrary sized grid, dependent on the desired precision. Each of the boxes in this grid is identified with a string. The longer the string, the higher-level resolution we will have.

For example, the latitude longitude pair of 39.5997° N, 105.0108° W would be bounded by the box 9XJ362VKZ. If we added more characters to this string and subdivided even further, we would be narrowing the area of the bounding box around this point. A geohash of only 12 characters would have a bounding box smaller than a meter [16], so we can be as precise as we can like and the geohash algorithm gives us a deterministic mechanism of data dispersion within the same group. An example of sub diving a geohash is demonstrated in figure 2.

This scheme, while giving us the desired effect of grouping data with similar features, leads the probability of storage imbalances across nodes in the system [5]. However, if we

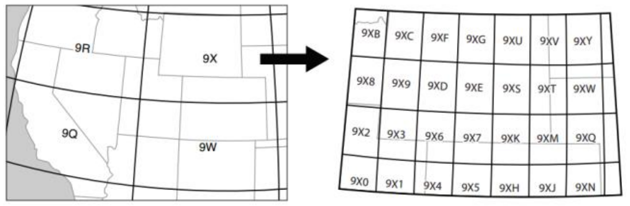

Figure 2. An illustration of the recursive subdivision of a geohash bounding box.

use a second hashing scheme for dispersion within the group chosen by the geohash, we can achieve a balance. We do this second tier hashing scheme on a feature of the data. While the geohashing algorithm is going to present imbalances in data placement within nodes, the dispersion between the geohash and SHA-1 algorithm is only 1% [5].

3) *Indexing: The Geoavailibilty Grid*

Indexing blocks in a distributed environment poses many interesting challenges. If we use a central index server, this quickly becomes our single point of failure and contact. However, if the index is shared across all nodes, this can lead to increased communication between nodes as they compare and update state involving the index among themselves. There are offline solutions such as R-trees[17] available, but these would leave a large memory footprint if they are indexing a billion files, not to mention the large amount of traffic that would be generated after every rebalance. Instead of these solutions, Galileo employs a Geoavailability grid for indexing. This grid translates points in space into a coarser resolution coordinate system. It is described by a vector of bits, or bitmap, where a bit is set to 1 if information has been stored in that location, and it is set to 0 otherwise.

Each node in the system keeps a Geoavailability grid of the location it is responsible for; however, any node in the system must be able to service any request to any node, regardless of failures in the system. Geoavailability grids are gossiped between all nodes. However, they are compressed before being sent and processed, and have the added bonus of not needing to be decompressed before being processed. [7]

After spatial information has been indexed in all of the Geoavailability grids, the system can then evaluate user-defined geospatial queries in the form of polygons encapsulating areas of interest. These polygons can be of any size, and are comprised of a list of latitude and longitude coordinates. These polygons are decomposed into smaller polygons if they are too large for current geohash resolution. For example, if our current resolution is 2 characters long, and this polygon's area covers two adjacent geohashes, it would be split into two different polygons, and treated as separate queries. These polygons are then consulted against the Geoavailability grids to find if data may be available and to eliminate nodes that cannot possibly service this query. Queries are also evaluated against other local metadata structures, the metadata graph and feature graph.

4) *Metadata and Information Retrieval*

Galileo allows users to query the system using features and feature values, rather than files or directories. This is accomplished using two data structures that each node contains in memory, a low-resolution feature graph detailing the global dataset, and a high-resolution metadata graph used for local evaluations of data stored on that node.

The low-resolution feature graph is used to reduce the overall search space when issuing a query. With this feature graph, queries can be issued to any node in the system, and it provides a coarse-grain view of all the dataset. This allows the node to have a global view of where to issue sub-queries throughout the system that will answer this query.

Nodes that receive this more directed sub query evaluate it against their local metadata graph. The high-resolution metadata graph instance at each node is populated with feature information that node is currently holding in files on the hard disk. It follows a hierarchical tree-like structure where each level of the tree corresponds to an indexed feature, and the leaves of the tree contain the data matching this path. Traversing this metadata graph from root to leaves allows nodes to narrow down queries to the relevant files that have these values along this certain path. However, traversing from leaf to root tells the node that this certain block has the feature values described by the path. This tree like structure allows us to group paths and sub-paths from root to leaf and hide duplicated data. This graph can be restructured or re-oriented, such that any query can be answered and the relevant data files matching this query are acquired, without ever having to go to disk.

The typical data flow involving these structures is as follows. A query is sent to any node in the system, and that node consults its respective feature graph to construct a set of candidate nodes to evaluate this query. This query is then forwarded to these nodes where they further evaluate the query on their local high-resolution metadata graph.

III. Distributed Evaluating Visual Query

GeoLens allows users to specify their query over raw datasets stored in Galileo. To support visual analytics over voluminous datasets in a timely fashion requires an efficient scheme to first retrieve, and then aggregate, and finally, navigate query outputs. To support these distinctive requirements, we evaluate visual analytics queries and process the corresponding query results on the storage nodes prior to delivery to the application. Our concurrent data processing allows the visual analytics application to interact directly with the dataset, rather than limit interactions to the pre-processed dataset.

A. *Self-adjustable Image Tiles and Data Locality*

As depicted in Figure 2, Galileo uses the Geohash algorithm to partition the dataset over multiple storage nodes. Each storage node is responsible for a set of geohashes; portions of the dataset corresponding belonging to the same geohash are stored on the same machine. Finer resolution image tiles are represented by finer resolution geohashes and this is naturally aligns data dispersion in Galileo. Therefore, data processing, such as aggregation, involves reduced data movements between storage nodes.

Adjusting the resolution of the image tile relies on the Geohash algorithm. Applications can increase this resolution by increasing the length of Geohash string. The longer the string, the smaller the size of the image tiles. With a Geohash string of 12 characters, the image tile represents 1 meter x 1 meter area. This enables the application to provide an effective minimum image tile size to the users.

The minimum image tile size is specified based on the maximum resolution supported by the device, and it is then translated to the corresponding Geohash resolution. When a user zooms-in to the image or zooms-out of the image, the image tile size and the Geohash resolution are re-calculated accordingly. For instance, if a user zooms-in to the image of the query results, the application increases the length of the Geohash string until it reaches the minimum image tile size. In contrast, if a user zooms-out of the image, the application merges the image tiles by reducing the length of the Geohash string.

B. *Visual Query Input*

We are faced with the need for geocoding, which is the process of translating a human readable name, such as Fort Collins, Colorado, United States, into its location on the earth. There are many geolocation services available such as geonames [18]. However, this is an online database and we do not want to rely on someone's network we have no control over. There are some offline solutions such as the NGA earth-info [19] but these are large files and require processing. Additionally, the output of these processes is a latitude and longitude pairs, denoting the center point of this area. We would need additional software to then find the series of latitude and longitude coordinates that bound this area.

To overcome this problem, and to receive a query area, we allow the user to draw their area they want directly on the map. GeoLens can also save polygons users have submitted so they do not need to be drawn again.

The features a user is interested should be included in the visual query. Galileo provides all of the features it is currently indexing, and the user can select all of the features they would like to examine. This polygon and the list of features are then sent to any node in Galileo to be examined.

C. *Distributed Query Evaluation*

As depicted in Figure 3, when a query arrives at any node, it follows the usual protocol of first consulting its Geoavailability grid and feature graphs to find potential groups that contain the information about this query.

This visual query is then distributed to all groups that have data that falls within this query area. This can be one group if that area is small, or multiple groups if the area is large, it depends on the size of the polygon and the volume of the data currently stored. The sub-polygons are generated, and these sub polygons are sent to the respective groups.

GeoLens instructs Galileo to select a visual query coordinator to evaluate a visual query. Most of the cases, the receiving node in the group plays the role of the coordinator. If the memory consumption and CPU utilization of the receiving node are higher than the configured thresholds, the receiving node pushes the request to the next node in the group. In Galileo, for a degree of replication N, there are N-1 other nodes in the group that are responsible for storing a copy of the same data item. This allows data locality to be preserved within the group.

The coordinator evaluates the user's query, orchestrates data aggregation and builds a *VisGraph* that is the representation of the visual query results. VisGraph is the set of all sub features such as heat maps and histograms. The coordinator issues the user's query to Galileo as if it is a regular Galileo query. The result of the query (a metadata graph) is used for planning further data processing. The coordinator retrieves the data locations from the result metadata graph and determines the node that will perform the partial data aggregation.

D. *Data Aggregation for Visual Analytics*

GeoLens provides several data reduction methods. Users can specify their own data reduction algorithms by using UDF (User Defined Function). Galileo has support for the sampling operator as a part of regular query.

The most popular data reduction techniques include filtering [20], sampling [21], and binned aggregation [22]. Filtering and sampling techniques are effectively used on the entire dataset to give us a more manageable subset in which to visualize. However, there is no native guarantee in the sampling algorithm that the sample we get is small enough to

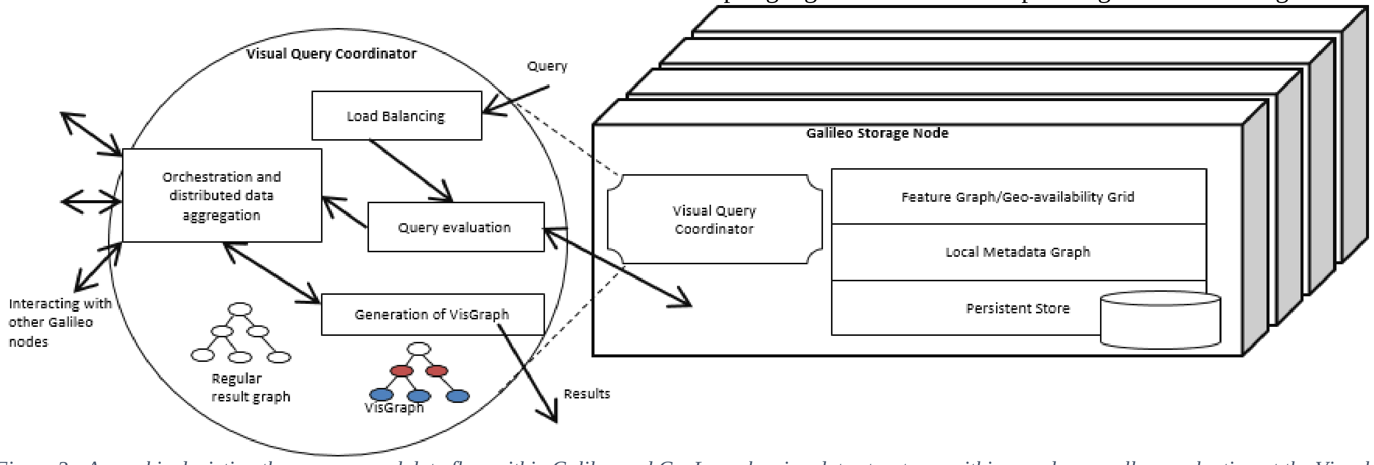

Figure 3 - A graphic depicting the message and data flow within Galileo and GeoLens showing data structures within a node, as well as evaluation at the Visual Query Coordinator.

fit within our visualization, so additional sampling might need to be done unless we are intelligent about our sampling. Another issue with sampling is it may omit outliers. Since every point has the same probability of being sampled, [21] we might omit important structures from the data simply because we got unlucky during our sampling. We can use some other form of sampling besides random sampling such as stratified sampling, or systematic sampling. However, these methods require prior knowledge of the structure of the data, and costly pre-processing.

Another data reduction technique is aggregation, or the creation of histograms describing the data. A histogram represents frequency distribution using rectangles whose widths represent class intervals and whose areas are proportional to the corresponding frequencies [8]. We support aggregation because it shows global patterns through spike height, and still preserves local outliers because it takes into account the entire data set. We include detailed information about our autonomous data aggregation scheme in Section IV.

E. *Creating the VisGraph*

A VisGraph is a metadata graph of the visual query results. As depicted in Figure 4, the Galileo metadata graph of a regular query contains pointers to the actual file location in its leaf nodes. The GeoLens coordinator transforms the metadata graph into a VisGraph. Unlike the metadata graph in a regular query result, the VisGraph includes aggregated values and histogram summaries that simplifies image rendering.

Visual analytics applications are required to render related images rapidly. For example, for a user who views and contrasts daily heat maps of temperatures in the US for March-2014, the application might need to render heat maps for any of the 31 days in March of 2014. To allow the users to traverse daily heat maps, the application will need to traverse among leaf nodes of the VisGraph. In general, the cost of traversal between the leaf nodes is $O(\log N)$, where N is the number of nodes in the VisGraph. To reduce the traversal cost, we organize the VisGraph based on the probability of leaf nodes being co-visited. The leaf nodes that have the highest chance to be visited together will share the same parent node. Finally, internal nodes closer to the root node indicate that traversal to those nodes is the result of a major feature change in the application.

The first step in creating a VisGraph from a metadata graph is to first split the paths based on date. Our separation technique is first to separate the data based on year, and then month, and then day of the month. Then, for every day of the month in every year, a histogram and geohash tile set are created for every feature. This is achieved by simply traversing the metadata graph and simply populating the new VisGraph with values that are in this particular month.

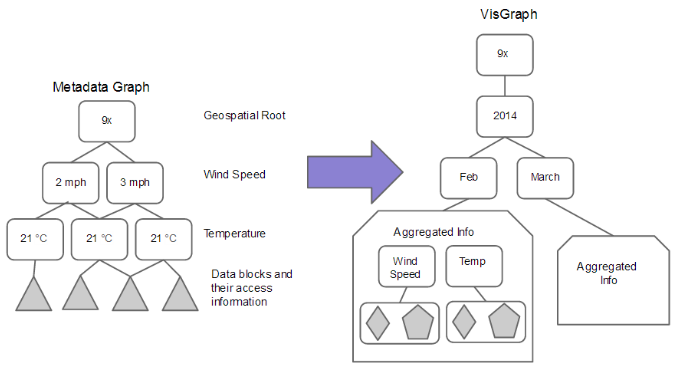

Figure 4 - An illustration of the transformation of a metadata graph to a VisGraph.

IV. Brushing and Linking

Brushing and linking combines different visualization methods to overcome shortcomings of using a single approach over large, complex dataset. In this paper, we have explored brushing and linking over heat maps and histograms of features using visual queries and VisGraph.

In order to enable brushing and linking on the client side, we created two different aggregations the data matched by the query. We aggregate the data in two different dimensions, and create two data structures sketching different facets of the data: feature histogram and heat maps

A. *Autonoumous Histogram Generation*

Aggregation, or binning, is the process of organizing the data into defined bins, and then counting each data point that falls into each bin. All of the features we examine in this paper are numeric, which leads to our aggregation technique being a set of uniform width bins, from the minimum value to the maximum value, for all of the features.

The biggest challenge here is bin width selection for the unknown dataset. The data is voluminous, and new data is continuously being added to the existing dataset. Users specify portions of the dataset to be visualized interactively. Information such as data distribution (including density of values) of the dataset is not known to the system.

There are a variety of ways we could obtain a bin width with which to aggregate data. A simple way would be to prompt the user. This would not be ideal because different features will have different widths, and the user might not know a good width for all of these features. In addition, the same features, but in different areas, could require different bin sizes. For example, an area that regularly experiences cold climates might not need the same guidelines as an area that experiences warm weather. Lastly, the same area might have different optimum bin widths, depending on the time of year. These reasons, coupled with the importance of the correct bin size described earlier, we do not leave it up to the user to supply our system with a value. Instead, we derive a bin width based on the data.

In order to create our histograms from the data, we need to establish two variables: the number of bins, and the width of those bins. The width of the bin is crucial to creating a histogram that best represents the data.

1) *Bin Size Prediction*

Our goal is to establish a histogram that best captures the underlying rate of our data. If we choose a width that is too wide, there are not enough bins to accurately portray the data. On the other hand, if the bin width is too narrow, we do not achieve the desired effect of reducing the data size. We need to find a width that both represents the peaks in our data, and gives us sufficient data reduction.

This issue has been addressed at smaller scales; however, when data is voluminous, the turnaround times for bin size prediction should be rapid enough so that it is applicable to interactive applications such as GeoLens.

Our goal is to achieve a bin size that provides approximately optimal width and also scales well with voluminous data stored in the distributed storage cluster. In order to find this algorithm, we compared and contrasted three different methods of obtaining the number of bars for a histogram: the N-Square method [9], the Freedman-Diaconis rule [10], and the optimal bin size algorithm from Shimazaki [11].

The N-Square method simply takes the bin size as $\sqrt{n}$, where *n* is the number of points in the dataset. This is the most efficient method, but does not consider any aspect of the data other than the size. The Freedman-Diaconis rule takes into account properties of the data when creating the histogram. It does not simply look at the size of the data, but also considers the distribution of the data. It does this by considering the interquartile range of the dataset, it is based off of Scott's normal reference rule [23], but is less sensitive to outliers because it uses the *IQR* rather than the standard deviation. By *IQR* we mean the interquartile range of the set.

The optimal histogram algorithm from Shimazaki attempts to calculate the peaks of the underlying data, and then picks a histogram that most resolves that peak. It does this by minimizing an estimated risk function. This algorithm finds the histogram that best describes the data, however, it is very expensive.

These algorithms are vastly different in how they arrive at the number of bars and their complexity. Figure 5 and 6 show our experiment on aforementioned approaches.

2) *Comparison of Histogram Generation Methods*

In order to decide which one of these generation methods we would use for our histogram creation, we looked at two different factors: their scalability with large data sizes, and the similarities in their output.

The optimal histogram generation algorithm picks a bar width that best describes the spikes in the underlying rate. It shows the smallest error between the underlying rate and its histogram. It gives the best value for the number of bars. The Freedman method does not do this, however it gives similar results, as depicted in Figure 5. The optimal histogram algorithm gives us the best results, however, we have

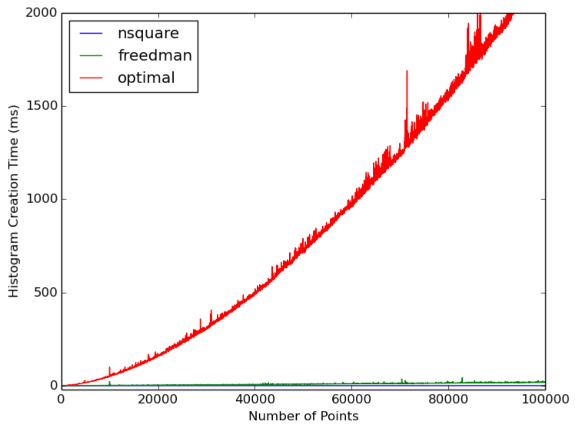

Figure 5 - The creation time of the three different methods. Even with only 5000 points, the scalability of the optimal histogram generation comes into question, but the other two methods scale.

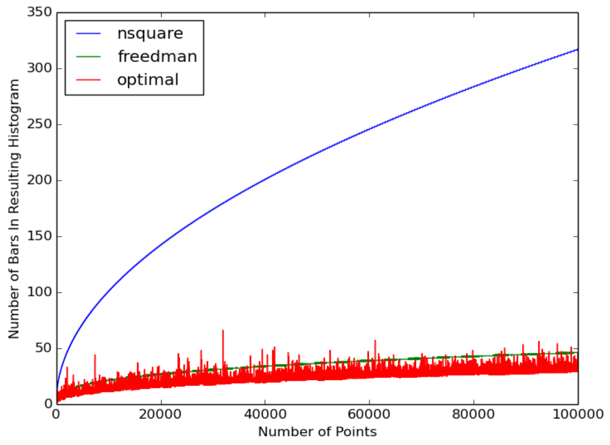

Figure 6 - The number of bars each method creates, dependent on the number size of the data. As the square root method soars off, the optimal and Freedman methods remain very similar.

discovered that this algorithm is not useable in the realm of big data. It is extremely expensive as shown in Figure 5.

One way we could mitigate the cost of this algorithm is by sampling from our larger dataset. However, this algorithm depends on three different variables, the minimum *min x*, the maximum *max x*, and the number of data points in the dataset *n*. In our sample, we would need to make sure to sample the maximum and minimum values, which would no longer make it a random sample. In addition, since this algorithm is dependent on the size of the data set, our sample can, and very likely will, generate a different bin size than it would if it worked on the entire dataset. Since this value is different than the optimal one for the entire dataset, it is no longer optimal.

The Freedman algorithm produced very similar bin sizes to the optimal bin-size algorithm. This is demonstrated in figure 6. Even though the Freedman does not pick the same bin-sizes as the optimal one, it oscillates around its values. That is why for its speed over time, coupled with the fact it chooses similar bin sizes, regardless of data size, we have chosen to use the Freedman-Diaconis rule as the method to autonomously generate histograms.

B. *Heat maps*

The geospatial aggregation process benefits from the strategy used in the self-adjustable data tile scheme. Merging and separating geohash boxes are done by adjusting the length of hash code string. Geospatial aggregation in heat maps requires minimum data transfers between the storage nodes because the data partitioning that happens in the storage system is closely aligned to the image tiles that will be displayed.

1) *Minimum Acceptable Geohash Size*

When a query is submitted into any node in Galileo, the geospatial coverage is specified as a form of a polygon. If this polygon covers more than one geohash area rectangle at the current resolution, the polygon is split into different polygons for evaluation at different nodes. After these sub-polygons are sent to the correct nodes, GeoLens harnesses this sub-polygon by first finding all of the geohashes that are inside of this polygon, and are two characters (configurable value) longer than the current global resolution. In this way we have a bounded size on the geospatial aspect of our geospatial visualization. It is bounded both in the max amount of geohash tiles we create, and is guaranteed to give us enough geohash boxes, without overloading the user's ability to perceive it.

2) *Creating the Geohash Dictionary*

After all the geohashes that reside inside of this polygon are found, GeoLens creates a dictionary with geohash values, and populates it with values from the data that are inside of this box. We already have a sketch of all the data values in this entire area surrounded by the polygon, so we need to create one for just the values inside of this polygon. This is done by averaging the values in this box, and reporting this average as the value for this geohash area. This creates a dictionary of the geohashes in this area, and a snapshot of those values in that area.

C. *Linking Between Heat Maps and Histograms*

Brushing and Linking requires fast interactions between different summarized views of the same dataset. When a user changes one of their views, the other view associated with the modified view should be immediately rendered. GeoLens links different views of dataset through VisGraph. Heat maps and histograms are linked by sharing ancestor nodes in a VisGraph.

V. RELATED WORKS.

A. *Traditional Techniques (Visualization Systems):*

There exist many tools available now for visualizing geospatial data. One such tool is the Integrated Data Viewer, published by Unidata [31]. This tool, like many other tools, streams data to a client from a database and plots every point. There is no upper bound on the amount on the size of the visualization, and no data reduction techniques. These tools also provide very vivid and rich visualizations, which take time to render. Scientists do not always want to wait on rich visualization. Sometimes they want to get a quick snapshot of the data, and to do this, there must be some aggregation.

B. *Web Based Analytics*

There exist a variety of general purpose tools for visual analytics, such as IBM's Many Eyes [24] and Google Fusion Tables [25]. These tools give anyone the ability to visualize data. However, these tools are not tailored to our specific needs. Our data is multivariate and contains various features, along with a geospatial location and time stamp. Also, to reduce data size, these tools use sampling, and visualize the sampled data rather than the entire dataset, thus incurring the sampling penalties descried earlier.

There are web-based tools that do aggregation such as Microsoft's Pivot [26]. This tools has a notion of a collection, and can aggregate collections along various dimensions that data would share. This allows the user to gain new insights by data around and allowing a user to look at certain bits and pieces at a time. But this tool is not tailored to our needs, because preprocessing is involved to visualize any data.

C. *Large Scale Visual Data Analytics*

There are existing products, such as AT&T's Nanocubes [27] or Stanford's imMens [28] that allow for visualization of large geospatial, time series, datasets, and allow for real-time queries on that data set. However, imMens requires users to predefine widths for aggregation. They do this to save time when creating data cubes, and because they have no reason not to, as the data they used did not have attributes besides a timestamp and geographic data. However, the data we desire to visualize contains an additional dimension as well as the geospatial time series aspect for each feature the user is interested in. For example, our data is not only about Colorado in October, it also contains the snowfall at that specific point, as well as any other feature involved. These different features require different bin sizes, and we do not expect our user to provide those values, as we can find a more precise value for them.

Our work is also strongly related to Nanocubes. However, Nanocubes is not suitable for the frequently updated dataset such as climate datasets. Since Nanocubes supports only read-only back-end data structure, a new nanocube would need to be created every time an update occurs. GeoLens is also capable of polygon queries covering any sized area, where as both Immens and Nanocubes can only support bounding box queries.

VI. Performance Evaluations

A. Experimental Configuration

For the data in this study, we used real world data from the National Oceanic and Atmospheric Administration (NOAA) North American Mesoscale Forecast System (NAM) [29]. Using Galileo's NetCDF plugin, we sampled from this dataset to create test data of one billion (1,000,000,000) files, each around 8 KB, spanning three months. The features we indexed included the spatial location of the samples, the time they were recorded, the surface temperature (Kelvin), total precipitation (meters), and wind speed (meters per second).

Galileo was executed in our heterogeneous 77-node cluster composed of 47 HP DL160 servers (Xeon E5620, 12 GB RAM, 15000 RPM Disk) and 28 Sun Microsystems SunFire X4100 servers (Opteron 254, 8 GB RAM, 10000 RPM Disk). However, only 13 of these HP nodes were utilized for these experiments. GeoLens was run on a single HP-Z220-XeonE3-12230 machine with 8GB RAM. The software providing the map of the world and the image tiles comprising the map were created by the OpenStreetMap community [30].

B. Performance Evaluation

In order to test the performance of GeoLens, we looked at the total time to create a snapshot for three different geographic areas. We then analyzed the time taken at each step of creating the VisGraph.

The steps taken into account are: the time to find points in the Geoavailability grid, the amount of time it takes to create the image tiles and histograms, and the length of time takes to display these on the client's computers. Finding the correct node in which to do the VisGraph creation on never takes more than two messages in the system, due to Galileo's zero-hop nature, and each node's feature graph. These values are averaged over 100 runs and shown in Figure 7. The values for searching the GeoGrid, as well as VisGraph creation, come from a single node within the network. The timer begins when a node gets a query, and the timer stops when that node sends its VisGraph to the coordinator to be merged. Different nodes were considered throughout the 100 runs. The final aspect, the time to display the visualization is the time to unmarshal the network representation of the VisGraph into the final image displayed on the users screen.

As this graph shows, GeoLens is extremely quick at giving a snapshot of the data, regardless of data size, or geographic size. A point of interest in this graph one might notice is the amount of time to locate data takes longer than in the larger cases.

Table 1 – Data Reduction in GeoLens

Original Size	Reduced Size
15.4688 MB	.684 KB
178.219 MB	1.806 KB
2316.84375 MB	3.736 KB

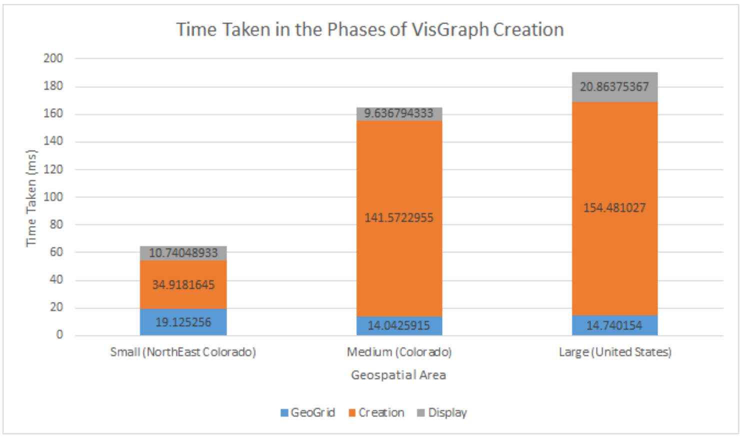

Figure 7 – The time taken during various phases of GeoLens.

This is because we are looking for points with a higher resolution in this case. We demonstrate the data reduction capabilities above, in Table 1.

These results show that GeoLens is not limited by the size of data or the geospatial area that is required.

VII. Conclusion And Future Work

A. Conclusions

Quick and effective visualization is a challenging problem when datasets are multivariate and voluminous, and providing brushing and linking capabilities with the visualized data provides additional challenges. For these issues, we have developed GeoLens, a framework built over an existing DHT framework, Galileo. GeoLens enables data search, retrieval, and aggregation on Galileo with sub-second response times to support interactive visual analytics. The system preserves data locality during visualization by means of aligning image tiling to the data partitioning (geohash resolutions) within the storage framework. We have incorporated brushing and linking into our distributed visual query scheme and also our data aggregation algorithms. Our autonomous histogram generation scheme is scalable and fast.

B. Future Work

When a part of a polygon is barely contained within a geohash, we visualize it in the same manner as if the entire geohash was covered. Instead, we could divide this one geohash box further, and get a more precise geo tile. This way, the visualization is more precise in the exact area it is showing.

Another area we could expand on is our autonomous histogram generation. Some of the features in the dataset we are using are categorical, rather than numerical, and if we try to aggregate these values using a numeric scheme, we are not going to have good results. Instead of creating a histogram spanning a series of numbers, we want to have counts of different categories. The research problem lies with deciding whether a set of numeric data is categorical or not.

Acknowledgements

This research has been supported by funding from the US Department of Homeland Security's Long Range program (HSHQDC-13-C-B0018).

References

[1] D. A. Keim, "Information Visualization and Visual Data Mining," *IEEE Transactions on Visualization and computer graphics*, 2002.

[2] R. Voigt, "An Extended Scatterplot Matrix and Case Studies in Information Visualization," Master's thesis, Hochschule Magdeburg-Stendal, 2002.

[3] Y. H. Beth Yost, and Chris North, "Beyond visual acuity: the perceptual scalability of information visualizations for large displays," *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07). ACM, New York, NY, USA,* , pp. 101-110, 2007.

[4] S. P. Matthew Malensek, and Shrideep Pallickara, "Geometry and Proximity Constrained Query Evaluations over Large Geospatial Datasets Using Distributed Hash Tables," *(To appear) IEEE Computing in Science and Engineering (CiSE). Special Issue on Extreme Data.*, 2014.

[5] S. P. Matthew Malensek, and Shrideep Pallickara, "Expressive Query Support for Multidimensional Data in Distributed Hash Tables," *Proceedings of the IEEE/ACM Conference on Utility and Cloud Computing, Chicago, USA. 2012.*, pp. 31-38, 2012.

[6] S. P. Matthew Malensek, and Shrideep Pallickara, "Galileo: A Framework for Distributed Storage of High-Throughput Data Streams," *Proceedings of the IEEE/ACM Conference on Utility and Cloud Computing. Melbourne, Australia.*, 2011.

[7] S. P. a. S. P. Matthew Malensek, "Polygon-Based Query Evaluation over Geospatial Data Using Distributed Hash Tables," presented at the Proceedings of the IEEE/ACM Conference on Utility and Cloud Computing, Dresden, Germany, 2013.

[8] D. Anderson, D. Sweeney, and T. Williams, *Essentials of Statistics for Business and Economics, Revised*: Cengage Learning, 2011.

[9] Microsoft. (2014). *Explore histograms: Office 2003*. Available: http://office.microsoft.com/en-us/excel-help/explore-histograms-HA001110948.aspx

[10] D. Freedman and P. Diaconis, "On the histogram as a density estimator:L 2 theory," *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete,* vol. 57, pp. 453-476, 1981/12/01 1981.

[11] H. Shimazaki and S. Shinomoto, "A method for selecting the bin size of a time histogram," *Neural computation,* vol. 19, pp. 1503-1527, 2007.

[12] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *Selected Areas in Communications, IEEE Journal on,* vol. 22, pp. 41-53, 2004.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review,* vol. 31, pp. 149-160, 2001.

[14] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin*, et al.*, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.,* vol. 41, pp. 205-220, 2007.

[15] R. Rew and G. Davis, "NetCDF: an interface for scientific data access," *Computer Graphics and Applications, IEEE,* vol. 10, pp. 76-82, 1990.

[16] W. Contributors. (2013, July 10). *Geohash*. Available: http://en.wikipedia.org/wiki/Geohash

[17] A. Guttman, *R-trees: a dynamic index structure for spatial searching* vol. 14: ACM, 1984.

[18] GeoNames. (Retrived 2014). *GeoNames*. Available: http://geonames.org/

[19] N. G. I. Agency. (Retrived July 2014). *NGA GEOnet Names Server (GNS)*. Available: http://earth-info.nga.mil/gns/html/

[20] S. B. Ahlberg C., "Visual information seeking: Tight coupling of dynamic query filters with starfield displays," *In Proceedings of CHI (1994),* , pp. 313-317, 1994.

[21] E. Bertini and G. Santucci, "Give chance a chance: modeling density to enhance scatter plot quality through random data sampling," *Information Visualization,* vol. 5, pp. 95-110, 2006.

[22] O. Rübel, K. Wu, H. Childs, J. Meredith, C. G. Geddes, E. Cormier-Michel*, et al.*, "High performance multivariate visual data exploration for extremely large data," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 2008, p. 51.

[23] D. W. Scott, "Scott's rule," *Wiley Interdisciplinary Reviews: Computational Statistics,* vol. 2, pp. 497-502, 2010.

[24] F. B. Viegas, M. Wattenberg, F. Van Ham, J. Kriss, and M. McKeon, "Manyeyes: a site for visualization at internet scale," *Visualization and Computer Graphics, IEEE Transactions on,* vol. 13, pp. 1121-1128, 2007.

[25] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley*, et al.*, "Google fusion tables: web-centered data management and collaboration," presented at the Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, Indianapolis, Indiana, USA, 2010.

[26] Microsoft. (2014). *PivotViewer*. Available: http://www.microsoft.com/silverlight/pivotviewer/

[27] L. Lins, J. T. Klosowski, and C. Scheidegger, "Nanocubes for real-time exploration of spatiotemporal datasets," *Visualization and Computer Graphics, IEEE Transactions on,* vol. 19, pp. 2456-2465, 2013.

[28] Z. Liu, B. Jiang, and J. Heer, "imMens: Real-time Visual Querying of Big Data," in *Computer Graphics Forum*, 2013, pp. 421-430.

[29] NOAA. (2013). *The NAMNOAA.* . Available: http://www.emc.ncep.noaa.gov/index.php?branch=NAM

[30] M. Haklay and P. Weber, "OpenStreetMap: User-Generated Street Maps," *Pervasive Computing, IEEE,* vol. 7, pp. 12-18, 2008.

[31] Murray, Don, et al. "13.2 THE INTEGRATED DATA VIEWER – A WEB-ENABLED APPLICATION FOR SCIENTIFIC ANALYSIS AND VISUALIZATION." (2003).