Aperture: Fast Visualizations Over Spatiotemporal Datasets

Kevin Bruhwiler
Department of Computer Science
Colorado State University
Fort Collins, USA
Kevin.Bruhwiler@rams.colostate.edu

Shrideep Pallickara
Department of Computer Science
Colorado State University
Fort Collins, USA
Shrideep.Pallickara@colostate.edu

ABSTRACT

One of the most powerful way to explore data is to visualize it. Visualizations underpin data wrangling, feature space explorations, and understanding the dynamics of phenomena. Here, we explore interactive visualizations of voluminous, spatiotemporal datasets. Our system, Aperture, makes novel use of data sketches to reconcile I/O overheads, in particular the speed differential across the memory hierarchy, and data volumes. Queries underpin several aspects of our methodology. This includes support for a diversity of queries that are aligned with the construction of visual artifacts, facilitating their effective evaluation over the server (distributed) backend, and generating speculative queries based on a user's exploration trajectory. Aperture includes support for different visual artifacts, animations, and multilinked views via scalable brushing-and-linking. Finally, we also explore issues in effective containerization to support visualization workloads. Our empirical benchmarks profile several aspects of visualization performance and demonstrate the suitability of our methodology.

Author Keywords

Spatiotemporal Data; Visualization; Sketching Algorithms; Containerization

CCS Concepts

•Information systems → **Spatiotemporal Data; •Human-centered computing** → **Visualization; •Theory of computation** → *Sketching Algorithms;* **•Software and its engineering** → *Containerization;*

1 Introduction

Spatiotemporal datasets proliferate in several domains. A large fraction of all datasets that are being generated are spatiotemporal. Domains that these datasets are available in include ecology, epidemiology, atmospheric sciences, geosciences, and commerce.

One of the most powerful ways to explore data is to visualize it. Visualizations are often a precursor to data wrangling, preprocessing, and subsequent analysis. However, the growth in cumulative data volumes alongside access latencies and low throughputs of the I/O subsystem pose formidable challenges during visualization. Visualizations also involve computational overheads relating to calculation of projections, tessellated polygons, and visual artifacts that need to be rendered.

The datasets we consider comprise a large number of observations and have several unique characteristics. Each observation is multidimensional and represented as a vector encapsulating features (e.g., temperature, humidity, etc.) of interest. All observations are geotagged. These geotags may either be a <latitude, longitude> coordinates or may represent a spatial extent. Observations also have a timestamps associated with them that reflect when the measurements were recorded. The data are encoded in a plethora of encoding formats such as netCDF, XML, etc. As data volumes have grown, it is infeasible to host the dataset on a single machine.

Visualizations entail network I/O, disk I/O, and computational overheads. As data volumes increase the strains placed on each of these phases also amplifies. The crux of this study is to enable near real-time visualizations of voluminous spatiotemporal datasets.

1.1 Challenges

There are several challenges in effective visualizations of voluminous, spatiotemporal datasets.

- Growth in data volumes represent an increase in the number of observations. This represents a corresponding increase in the number of observations that must be sifted through, processed, and rendered.
- Characteristics of the memory hierarchy. Datasets are resident on disk and portions thereof must be memory-resident for visualization. However, access latencies, capacities, and data transfer throughputs are steep across the memory hierarchy. For example, memory accesses are 5-6 orders of magnitude faster than those to disk.
- Fast response times: For explorations to be effective visualizations must be interactive.
- Network I/O: Data must be transferred over the network from the server-side for visualization.

Data volumes exacerbate these challenges.

1.2 Research Questions

The broader goal of this study is interactive visualization of voluminous spatiotemporal datasets. Specific research questions that we explore as part of this study include:

RQ-1: *How can we cope with data volumes and the speed differential of the memory hierarchy?*

ISBN .
DOI:

RQ-2: *How can we effectively orchestrate workloads associated with visualization?*

RQ-3: *How can we effectively facilitate multi-linked views?*

RQ-4: *How can we preserve interactivity during visualization?*

RQ-5: *How can we leverage containers in support of effective visualizations in private/public clouds?*

1.3 Approach Summary

Our methodology leverages a novel mix of sketching, construction of queries, visual artifacts, orchestration of workloads, management of workloads, and caching, all of which work in concert with each other to support effective visualizations.

To cope with data volume, we first construct sketches of the data using the Synopsis sketching algorithm. The Synopsis[4] sketch is a space-efficient representation of the data space. These sketches are an effective surrogate for the data and reconcile diverse data encoding formats. In fact, the sketch is a compact, encoding-agnostic representation of the data space. The sketch is distributed and a 1000-fold more compact than the data. The compactness of the sketch allows us to pin the entire sketch (or large portions thereof) in memory allowing us to reduce the amount of disk I/O that need to be performed. Once constructed it is the sketch, not the on-disk data, that is consulted during visualization.

Visualization represents a sifting of the observational space to render items of interest. We leverage specially calibration queries to accomplish this. Our queries are aligned with the needs of the visual artifacts that need to be constructed and rendered. Support for queries and retrieval of information from the sketches.

Workloads for visualization includes a mix of client-side and server-side workloads that must be effectively orchestrated.

Caching is utilized on both the client and the server to increase responsiveness and enable interactivity. The server maintains a large LRU (least recently used) cache of data and image rasters, which it uses to increase the interactivity of certain query types and allow users to quickly return to previously queried data. The client maintains a much smaller LRU cache of query outputs that it predicts the user will want to view in the near future.

The Synopsis sketch is parsed into data pairs consisting of a geohash and the corresponding feature data. The geohash is then decoded into a latitude/longitude point and passed through a Mercator projection so that it conforms to the tile-map we use as a background. The points are projected a second time to centre them within the queried bounds.

The Delaunay triangulation[8] of these points is computed and used to generate a Voronoi diagram consisting of a set of tessellated polygons, each centered on one of the projected latitude/longitude pairs. This set of polygons defines the regions of our choropleth map. The choropleth map is rendered by filling in the regions with colors based on the feature data corresponding to the point at the center of the polygon.

To support multilinked views we leverage brushing-and-linking. In multilinked views, selection of data items within one view (say a Choropleth) triggers a selection and rendering in another view (say a histogram). To support brushing-and-linking we decompose this into two distinct, but related queries that are used to simultaneously generate statistical summaries, and perform sifting operations to identify the subset of observations that must be rendered.

We also leverage a user's navigational trajectory to ensure interactivity during visualization operations. In particular, we speculate a user's likely exploration trajectory (in the immediate spatial or temporal vicinity of the current visualization) to launch speculative tasks that precompute visual artifacts. Some of these speculative tasks may not materialize; however, our methodology frugally launches speculative tasks so that the number of materialized visualization artifacts that must be discarded our kept at a minimum.

Our benchmarks demonstrate the suitability of our methodology. The experiments were performed with well-known voluminous, multidimensional datasets.

1.4 Paper Contributions

In this study, we present a methodology for interactive visualization of voluminous spatiotemporal datasets. Our methodology combines several innovations to accomplish this goal. In particular, this includes:

1. Effective distributed orchestration of workloads (CPU, disk I/O, and network I/O).
2. A novel scheme to visualize datasets by leveraging sketches. To our knowledge, this is the first attempt to leverage spatiotemporal sketches in support of real time visualizations.
3. Support for interactive, multilinked views over voluminous spatiotemporal datasets. In particular, we leverage the sketch to support cross-referenced rendering and visualization at scale.
4. We can support animations of voluminous spatiotemporal phenomena at two frames/second.
5. We have designed a scheme to effectively containerize sketches. These distributed containerized-sketches are in the critical path of an interactive visualization application with stringent latency requirements.

2 Related Work

2.1 Visualization Tools

There are many existing tools for data visualization designed to function with a variety of different data and storage systems. Data Cubes[21] are a classic tool for visualizing multi-dimensional data in relational databases and owe their popularity to both the prevalence of relational databases and their ability to handle any data type. However, unlike Aperture, they don't perform any data aggregation, making them prohibitively expensive to compute and store for voluminous datasets. Data Cubes are also confined to the relational database ecosystem, making them unsuitable for certain datasets.

Tableau[22] is a powerful data visualization tool specializing in business intelligence. It's usefulness derives from its ability to handle data from many different sources and from its ease of use, being accessible to non-programmers. Like Aperture,

it contains tools for spatiotemporal aggregations and visualizations. However, Tableau was not designed with interactive queries in mind and the time required to query, format, and display large amounts of data makes animations and interactivity over arbitrarily large datasets infeasible.

GeoLens[14] is a distributed geospatial data visualization tool designed to allow interactive visualizations of aggregated spatiotemporal data and consequently shares many similarities with Aperture. Both query data stored on a cluster of machines, both implement brushing and linking, and both aggregate spatiotemporal data using geohashes. However, GeoLens utilizes a sophisticated distributed query scheme to rapidly aggregate data on each machine into tiles, combining the tiles to produce a fast, low-resolution visualization. Aperture, alternatively, performs individual queries on a single machine, allowing it to produce much higher-resolution visualizations at a slower pace. Aperture compensates for the speed difference with speculative queries and a server-client caching scheme.

2.2 Data Sketching

Data sketching defines a set of techniques which build a probabilistic approximation of a dataset, dramatically reducing the dataset's size at the cost of some precision. Such algorithms can dramatically reduce storage requirements and query time (essential for interactive visualizations) and can even be used for machine learning algorithms[13]. Aperture makes use of a distributed sketching algorithm called Synopsis[4] (see **section 3.1**) which is designed for geospatial data, however a number of alternatives exist including algorithms which focus on processing multiple input streams[10], anomaly detection[12], and threshold monitoring tasks[9].

2.3 Geospatial Data Storage

Aperture relies on Synopsis to store geospatial data and process spatiotemporal queries, however there are a number of other systems that provide similar functionality. These systems could conceivably replace Synopsis as a geospatial data store for Aperture. However, since Synopsis is a primarily in-memory storage system, it's also sensible to view alternatives as stable storage systems which Synopsis itself could query data from to create a sketch for Aperture.

MongoDB[6] is a very popular database with geospatial storage capabilities. It supports querying ranges of latitude and longitude, as well as queries that calculate geometries on an earth-like sphere. <latitude, longitude> coordinate data can be easily converted into a sketch by Synopsis, making MongoDB a natural choice for an underlying database.

Galileo[16] is a distributed data storage system designed with geospatial data in mind. It is organized as a distributed hash table (DHT) using a geohash-based hashing scheme. It also implements zero-hop indexing, ensuring that retrieving data for a specific region is very fast. Converting data stored by Galileo into a Synopsis sketch is trivial, making Galileo another good choice for an underlying database.

Stash[19] is a system which operates on on-disk data and supports selective caching and memory-evictions during explorations. In particular, freshness scores are used to inform cache evictions. The system relies on re-replications and migrations to cope with hotspots introduced by disk accesses. Unlike Stash, we support multilinked views, leverage sketches, and perform speculative queries in support of interactive visualizations.

2.4 In-Memory Data Storage

Aperture owes much of its speed to the in-memory nature of Synopsis. There are, however, a number of other in-memory data storage systems that could potentially replace Synopsis in Aperture.

Redis[5] is an in-memory distributed data store which supports most data types, many programming languages, and a number of typical database properties, including replication and atomic operations. It also supports indexes and even allows querying by geohash. However, Redis does not aggregate geospatial data. This would force Aperture to do the aggregation itself and significantly reduce the amount of data that could be stored in-memory and quickly visualized.

Apache Ignite[2], an in-memory distributed data store , shares many features with Redis. It comes with a geospatial library which allows for sophisticated queries on points, lines, and polygons. Unfortunately, Ignite also does not natively support spatial data aggregation and consequently shares Redis' limitations. The lack of spatial data aggregation in Redis and Apache Ignite could conceivably be addressed by pre-sketching the data before loading it into the data store, however this would require the development of a new system which can aggregate spatiotemporal data in a manner easily understood by Ignite or Redis.

Nanocubes[15] are a modification to the Data Cube system which dramatically reduce the amount of data that needs to be visualized using aggregation. This allows Nanocubes to be memory-resident, enabling them to evaluate queries at exceptional speeds. However, unlike Synopsis, Nanocubes cannot be distributed across multiple machines, limiting the amount of data they can be used to construct visualizations to a single machine's memory. This issue could be dealt with by maintaining many separate Nanocubes on different machines, similar to what we do with Synopsis and Docker[18] in **section 3.6**.

3 Methodology

3.1 Leveraging Sketches for Visualization [RQ-1]

Visualization of spatiotemporal datasets is I/O bound. As data volumes increase the I/O overheads preclude interactivity. To address these data and I/O challenges, we leverage the Synopsis algorithm to sketch voluminous geospatial data. Synopsis makes a single pass through all records within a dataset (either streaming or on-disk) to produce a distributed, space-efficient representation of the spatiotemporal data. Once the sketch is constructed, it is the sketch rather than the data that is consulted. Synopsis makes use of the geohash algorithm (figure 1) to partition and agglomerate spatiotemporal observations.

The Synopsis sketch is organized as a forest of trees, which maintains summary information and distributional characteristics that are updated in an online fashion. Synopsis maintains its underlying representation as a distributed, memory-resident tree, scaling it across machines as necessary. The organiza-

tional structure of the sketch allows for rapid geospatial queries and aggregation. Synopsis is crucial to Aperture's interactive nature as it allows the user to perform aggregate queries with sub-second latencies and dramatically reduces the scale of the data the client must load and store.

The geohash algorithm (figure 1) encodes geospatial coordinates into a bit array, which is then represented as a string. A geohash string represents a spatial bounding box encompassing a unique geographical extent - all coordinates within that extent share the same geohash. The geohash string has the configurable precision property: reducing the length of the string increases the spatial area of the region identified, analogous to "zooming-out". Conversely, increasing the number of characters in the string can be thought of as "zooming-in" to an increasingly precise location.

Table 1 shows the geographical extent of different geohash precisions for locations near the equator, as well as a heuristic for the region that the extent covers

Geohash Precision	Geographical Extent	Scale
1	5,009.4km x 4,992.6km	Continent
2	1,252.3km x 624.1km	State
3	156.5km x 156km	County
4	39.1km x 19.5km	City
5	4.9km x 4.9km	Block

Table 1: Geographical extent of different geohash precisions, measured in number of characters, located near the equator.

Visualization is underpinned by queries over the observational space, and the efficiency of these evaluation is vital. We conducted microbenchmarks to assess the effectiveness of Aperture with and without Synopsis at the backend. We profiled the performance of Aperture's data queries while sidestepping Synopsis and reading data directly from the hard drive. The comparative performance is depicted in figure 2. The rendering precision used was five characters. As can be seen, direct queries to disk require more than 14 seconds to complete compared to sub-second latencies that are made possible by leveraging the Synopsis sketch. Furthermore, disk-based

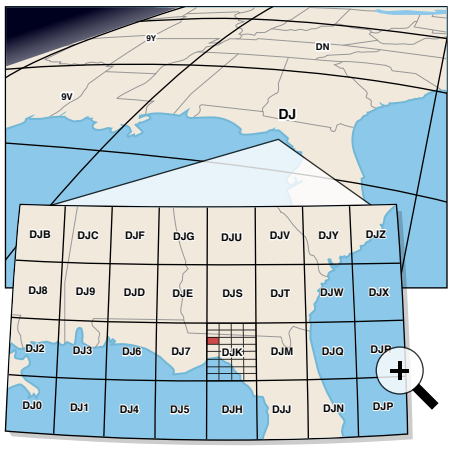

Figure 1: The geohash algorithm. Adding characters to the right side of the string reduces the area covered by the geohash and represents a more precise location.

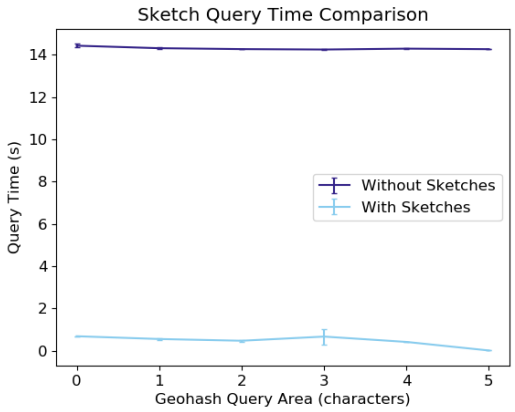

Figure 2: A comparison of the time Aperture takes to complete a query with and without Synopsis. Performing queries on sketches is more than 14 times faster than queries on un-sketched data.

queries generated nearly 3.5 gigabytes of network traffic that must occur on shared clusters and will degrade performance for other, colocated applications. **A comparison of rendering precisions was impossible because the amount of data returned by the direct-to-disk queries invariably crashed the Aperture client.**

Also, not captured in figure 2 is the fact that only a single query is being performed on the disk. Aperture's speed and interactivity depend critically on being able to run many queries simultaneously in order to both pre-load data that may be viewed in the future and load statistics and masks for the data currently being viewed. Attempting to run multiple simultaneous queries on disk would produce an enormous amount of I/O contention, cause throughputs to plummet, and likely force the queries to be performed serially. This would dramatically reduce the responsiveness of Aperture and make interactivity impossible.

3.2 Rendering Spatiotemporal Phenomena [RQ-2]

Users interact with Aperture via a web page which displays the data over a pannable and zoomable choropleth map of the world (figure 3). Users are able to specify their queries in terms of spatiotemporal regions and can filter the results based on the type and the minimum/maximum values of the data features they wish to view. The precision at which the data is rendered is also configurable. Once generated, the query is handled by the server.

Aperture allows users to automatically advance their visualizations through time at a specified number of frames per second. These visual artifacts form an animation which can be used to explore data over a temporal range. Animations are also zoomable and pannable while being played. Consequently, Aperture allows for exploration and analysis of both spatial regions and temporal regions *simultaneously*.

Aperture also includes two visualizations of the query's aggregated statistics (figure 4): a correlation matrix illustrating the correlation of each feature with each other and a histogram showing the distribution of values for each feature. These visualizations are linked to both each other and the choropleth map through a system of brushing and linking; zooming and

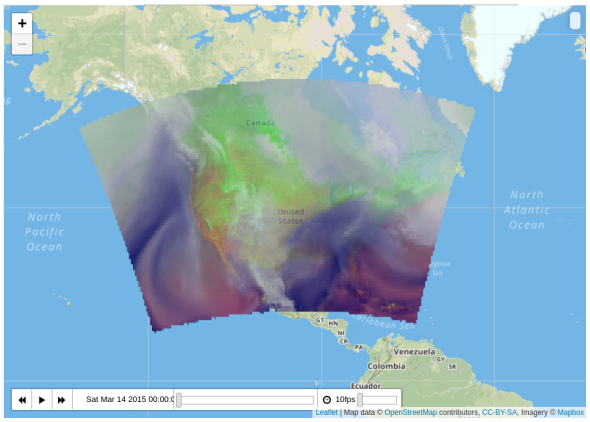

Figure 3: Aperture's choropleth map. The map consists of many tessellated polygons, each colored to represent measurements within their area. Here temperature (red-green), humidity (white), and precipitable water (purple) are shown.

panning across the choropleth map will update the aggregated statistics with the data being viewed while interacting with either of the statistical displays will update the other statistical display and highlight the appropriate part of the choropleth map.

Aperture leverages two key libraries to render spatiotemporal phenomena - Leaflet and D3. Leaflet[7] is an open-source JavaScript library that provides utilities for displaying spatiotemporal data. It supports functionality for tile-layers and SVG layers, is mobile-friendly, and contains many utilities that make it easily modifiable and extendable.

D3[3] is a JavaScript library that allows for efficient manipulation of data and Document Object Model (DOM) elements on a web page. It is frequently used for scientific data visualizations and gives an enormous amount of flexibility to developers to create dynamic displays.

Aperture utilizes Leaflet for two purposes. Firstly, it is used in conjunction with MapBox[17] to display a zoomable tilemap of the Earth, giving users information about where the displayed data is in relation to cities and coastlines. Secondly, Leaflet's layer functionality is used to display a rendered choropleth map generated by Aperture. Leaflet provides tools to zoom into and pan across the rendered image as well as utilities to calculate the location of the user's pointer and the area of the image currently viewed. These tools allow Aperture to have a higher level of interactivity than if the data were rendered statically. Aperture also makes use of Leaflet Time Dimension layers[20], developed by the Balearic Island Coastal Observing and Forecasting System (SOCIB), to assist in displaying animations over temporal ranges.

Aperture uses D3 to create and modify the statistical displays on the client. Fast modification of all the statistical displays during user interactions (brushing and linking) is critical, as even short delays in response time can cause frustration and break a chain of thought[1]. This makes D3 an integral aspect of Aperture's functionality.

3.4 Scalable Brushing and Linking [RQ-3]

A key feature supported by Aperture is multi-linked views at scale. Multilinked views are a method of visual exploration in which several corresponding aspects of the data are visualized simultaneously. Brushing and linking describes a system in which modifications to one view of the data (panning/zooming/mousing-over/clicking/etc.) trigger equivalent modifications in every other view. For example, narrowing the geographical scope of one view will cause the other views to narrow their scope accordingly.

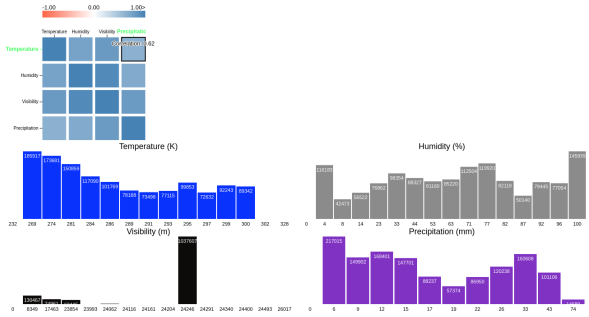

Figure 4: Aperture's statistical views. The matrix shows the correlation score between every pair of features. The bar charts show the distribution of measurements for each feature.

In Aperture, brushing and linking is accomplished by visualizing the results of two different types of queries: mask queries and statistical queries. Both of these queries are described in detail in **sections 3.5.2** and **3.5.3**, however in brief: statistical queries retrieve metadata about a particular geospatial region of the data while mask queries retrieve geospatial regions matching given the particular metadata constraints.

Interacting with either of the statistical visualizations will prompt a mask query, which is used to highlight all of the geospatial regions which meet the requirements of the selected statistic. For example, if the correlation between temperature and humidity is selected then the mask query will return all the geospatial regions for which the correlation is equivalent or greater. Alternatively, if a temperature range is selected then the mask query will return all the geospatial regions where the temperature falls within that range. The result of the query will be used to highlight the appropriate regions on the choropleth map and update the other statistical displays.

Conversely, interacting with the choropleth map prompts a statistical query, which is used to retrieve statistical information about a single geospatial region. Interactions include anything that changes the data overlaid on the map, primarily zooming and panning. This ensures that the statistical displays will always show an aggregation of the data currently visible on the choropleth map.

3.4.2 Speculative Queries [RQ-1, RQ-4]

To ensure interactivity during visualizations our methodology includes support for launching speculative tasks. These speculative tasks initiate actions based on forecasts of a user's likely trajectory for explorations. By performing such tasks in the background we reduce the amount of work that needs to be performed in the critical path of explorations.

Based on a client's exploration paths we dynamically generate speculative queries at the client-side. These speculative queries are generated in two cases:

1. While an animation is being played

2. During a pan crossing more than two geospatial regions

In the first case, Aperture proactively sends query requests to the server for times that may become part of the animation in the future. Data for times that will be viewed soon can be cached on the server and buffered on the client, ensuring that the animation proceeds smoothly.

In the second case, Aperture keeps track of the user's panning trajectory. Sequential requests to adjacent locations will trigger the web page to query locations further along the user's current trajectory. The results of these queries are stored in the server cache and in a client-side buffer, reducing loading times when visualizing many adjacent geospatial areas.

3.5 Server-side Orchestration of Visualization Workloads

The server is responsible for handling all queries, a process which includes querying the distributed memory-resident Synopsis sketch, rastering the output of the queries, and caching the results for future use. The queries take three forms: data queries, mask queries, and statistical queries. All queries are handled by a server utilizing a dynamic thread pool. This allows the server's operations to be extended to a machine with many cores or a cluster of machines commensurate with the needs of the system.

3.5.1 Data Queries [RQ-1, RQ-2]

Data queries are the core of enabling Aperture functionality. They are performed whenever the user changes either the features being viewed, the spatiotemporal scope being queried, or the precision at which the data is rendered. Data queries consist of four distinct phases: query, decoding, computation, and rendering, all of which are described below.

Query Phase: In the query phase the server converts the query created by the web page into a query that can be understood by Synopsis. The output from Synopsis is a serialized version of the portions of the sketch which satisfy the query.

Decoding Phase: In the decoding phase the sketch is deserialized. The geohash corresponding to each data point in the sketch is decoded into its latitude and longitude values and stored along with the feature data.

Computation Phase: In the computation phase a Mercator projection is applied to the decoded latitudes and longitudes which are then used to compute a Voronoi diagram. This Voronoi diagram consists of a set of tessellated polygons, each one corresponding to a single geohash.

Rendering Phase: The polygons created in the computation phase are rendered as a choropleth map, the colors of their regions corresponding to the feature values for each geohash. The map is then converted into a raster which is sent to the client.

After all these phases are complete the server caches the data created during the decoding phase, the geohash-polygon relationships created during the computation phase, and the raster created during the rendering phase.

3.5.2 Mask Queries [RQ-2, RQ-3]

When a portion of one of the statistical displays is selected, the Aperture client-side library iterates over the queried data to determine which geohashes meet the statistical requirements. These geohashes are sent to the server, which uses the cached geohash-polygon relationship to render an image which the choropleth map will use as a mask. If many data queries have been run in between a mask query and its associated data query then the polygons may have been dropped from the cache and the Voronoi diagram will need to be recomputed. The rastered image is then returned to the web server, which combines it pixel-by-pixel with any other active masks to create the new choropleth map to be rendered by Leaflet.

It is also the responsibility of the server to compute the updated statistics for the masked region. It should be noted that these statistical calculations cannot depend solely on Synopsis queries, as Synopsis only aggregates metadata across spatial regions, not statistical ones.

3.5.3 Statistical Queries [RQ-3, RQ-4]

Aperture leverages the spatial characteristics of the underlying sketch to perform rapid statistical spatiotemporal queries. Both the correlation and the distribution of features displayed over the map are shown as a matrix and bar chart visualization respectively. These statistics are retrieved by metadata queries, which rapidly aggregate the data contained within a provided geohash. The correlations and distributions are recomputed for every map pan and zoom to reflect the portion of the data that is currently being viewed.

We determine the exact region to query by calculating the geohash of the upper left corner and lower right corner of the visible region to a precision of 12 characters. Afterwards, the smallest geohash that contains both corners is computed and used as an estimate of the currently viewed area.

Since information about distributional characteristics are maintained in the sketch the overheads associated with statistical query evaluations tend to be low. This allows the statistical views to be responsive (interactive) to both drill-down/roll-up and panning operations. The statistical displays are constantly changing to reflect the spatiotemporal scope of the data being viewed on the choropleth map, allowing users to rapidly identify and quantify areas of particular interest.

3.6 Leveraging Containers to Orchestrate Server-side Workloads [RQ-5]

We explored how to leverage core cloud constructs to orchestrate visualization workloads. In particular, we viewed containers as a promising avenue. Containers facilitate lightweight packaging of a program together with its necessary dependencies and data. Furthermore, containers allow an application to be easily moved from one computing environment to another or replicated across a cluster of machines. This makes containers a natural fit for distributed server-side applications as the amount of resources the application consumes can be quickly scaled in or out to match user demand.

However, there are some difficulties in implementing a container-based server for applications that require access to voluminous data. Packaging large amounts of data into containers makes them unwieldy, increasing the time and resource cost of scaling services. Additionally, re-reading large chunks of data every time a container is moved or a service is replicated is slow and will significantly escalate disk I/O

contention in busy data centers. In this study, we leveraged Docker[18] to quickly create and deploy applications in containers. We also investigate the benefits of utilizing sketched data in containerized applications.

Aperture's Docker-based server works in the same manner as its basic server with the same query types, caching scheme, and data query phases. We create a separate Docker image for data from each day in the dataset to allow the container-orchestration system to dynamically replicate containers containing days which are receiving a disproportionate number of queries.

Upon start-up, all containers contact a central server with the container's location and the date of the data it contains. When the client wishes to query a specific date, it first contacts the central server to get the location of a relevant container before sending the query directly to the container. This allows the container-orchestration system to place containers wherever it chooses without worrying about the client.

A challenge that we encountered is that the data for a single day in our dataset totals more than 13.45 GB. Creating a sketch from that data requires 4 minutes and 10 seconds. This wait-time to scale up a service is untenable, especially in an interactive environment where consistent slowdowns prove extremely frustrating for users, and these measures fail to consider the time required by the container-orchestration system to load/move the huge image to a specific machine.

To tackle this, we pre-sketch the data and package a serialized version of the sketch into the container. As a result, images containing sketched data are an order of magnitude smaller and faster to start-up than their un-sketched counterparts, allowing Aperture's Docker-based server to scale more dynamically and consume far fewer resources than it otherwise would. It should also be noted that we did not compress the sketched data. The size of the serialized sketch could be further reduced in a resource constrained environment using any typical compression algorithm at the cost of increased start-up time due to decompression.

4 Performance Evaluation

To evaluate the effectiveness of our methodology we profile the performance of each individual query and of the system as a whole.

The queries are profiled based on two metrics: time and network traffic. Fast response times are essential for an interactive system as long wait times quickly frustrate users or break their train of thought. Network traffic is less critical, however systems that require high bandwidth will suffer serious performance issues in resource constrained environments. It is also a useful measure of the degree to which Aperture reduces voluminous data to a manageable level.

Query metrics are compared across two controlled variables. The first is the query area, measured in the number of characters in the geohash. The query area is a measure of the size of the area being queried. The second is render precision, again measured by the number of characters in the geohash. The render precision is a measure of the precision of the returned data; a query with higher precision will return a choropleth map with more regions.

4.1 Data Query Profile [RQ-1, RQ-2]

Data queries are the core of Aperture, and as such their performance is paramount. In figure 5 we report the time required for each phase based on the area being queried. From figure 5, it is clear that the decoding phase takes up the bulk of the time for queries over large areas (more than every other phase combined). The time required to send the results over the network is also significant. It is also worth noting that Synopsis' query time remains relatively constant regardless of the size of the query area.

In figure 6 it can be seen that the rendering precision has a negligible effect on query time at low precisions. It's only at higher rendering precisions that the *Computation* and *Rendering* phases begin to take up a significant amount of query time. However, due to the fact that increasing the rendering precision by one character can result in the generation of 4-8 times as many polygons, it is likely that the *Rendering* and *Computation* phases will dominate the query time at higher precisions.

Each data query generates a minimum of 5MB of network traffic with a small query area and a maximum of 20 MB with a query area of zero characters. This traffic consists almost entirely of the response to the client, which contains a rendered image and a substantial amount of metadata.

4.2 Mask Query Profile [RQ-3, RQ-4]

Mask queries are highly dependent on data queries, making them difficult to profile. Mask queries utilize the set of polygons generated by the data query, and consequently if the polygons have not been cached or have been evicted from the cache in the time since the corresponding data query has been executed, the data query must be re-executed.

The performance of mask queries is also dependent not only on the area being queried and the resolution at which the data is displayed but also the area of the map that must be masked. When quantifying the area covered by the mask, a query area of zero characters and a render precision of five characters is used as a standard. Consequently, the profile given here can be

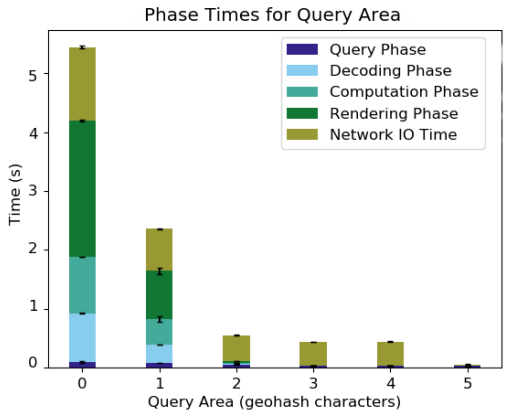

Figure 5: A comparison of data query times, separated by query phase, over different query areas. Network and query phase time are relatively constant, while the decoding, computation, and rendering phases dominate query time for large areas.

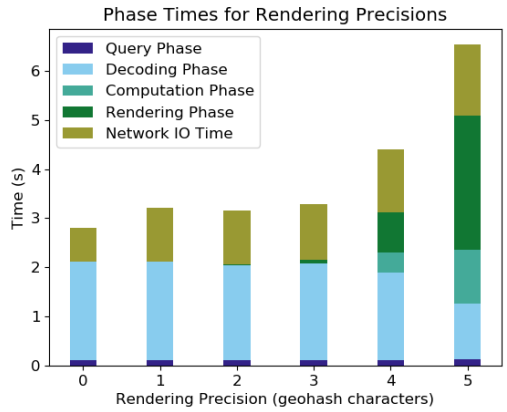

Figure 6: A comparison of data query times, separated by query phase, for different rendering precisions. At low values the precision is largely irrelevant, but it significantly increases the rendering and computation phase times at high values.

viewed as the performance of mask queries under the highest possible stress.

Under these conditions mask queries generate between 4 and 4.5 MB of network traffic, increasing with the amount of the current view being covered. Mask queries without caching enabled require the associated data query to be re-executed, which dominates the mask query time. In this case, the mask query and its data query require six seconds to execute regardless of the area covered.

Performance with caching is considerably improved (figure 7). Mask queries take approximately one-and-a-half seconds to complete, increasingly only slightly with the area being covered.

Responsiveness is essential for mask queries, as they are responsible for interactively linking the choropleth map with the statistical information. This profile illustrates how essential the server cache is to ensuring an interactive system.

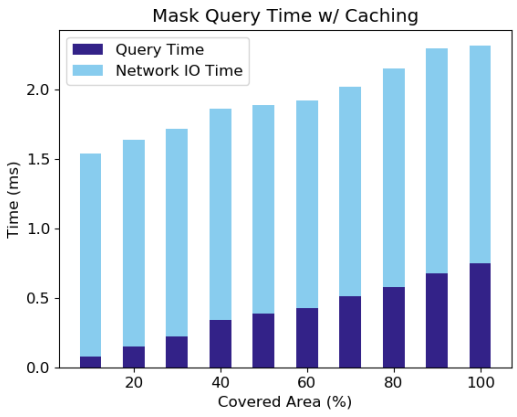

Figure 7: The query and network time required to perform a mask query based on the area of the current view the mask covers with server caching enabled. The covered area increases the query time slightly, with about a half-second increase from 0% to 100% coverage.

4.3 Statistical Query Profile [RQ-3, RQ-4]

Statistical queries are easier to profile. The network traffic they generate is invariant and never more than 2 KB per query. Statistical queries are also agnostic to the rendering precision as they simply display an aggregate of all the data within the queried bounds regardless of the number of regions in the choropleth map.

The time required for statistical queries is independent of the area being queried, requiring less than $1/5^{th}$ of a second of a second. Statistical queries generate the same amount of network traffic and require roughly the same amount of time to complete regardless of the scope and precision of the given query. This is due to the ease with which Synopsis aggregates data across spatial regions and is responsible for one of Aperture's greatest strengths: the interactivity of statistical information during drill-down, roll-up and panning operations.

4.5 Total Performance Profile [RQ-1, RQ-2]

Finally, the performance of Aperture is profiled across two real-world tasks. The first is an animation through time in

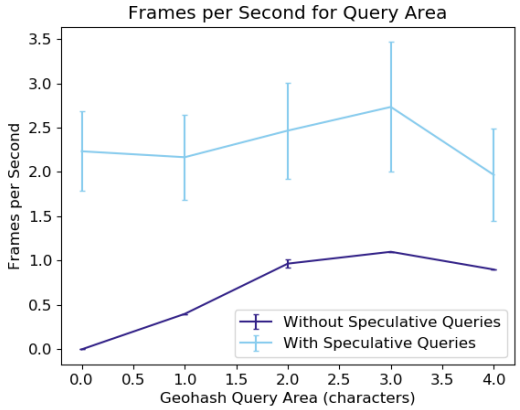

Figure 8: The number of frames in an animation that Aperture can display each second depending on the query area. Speculative queries have a major impact when the query area is relatively large, more than doubling the animation speed. Each frame consists of 3gb of unsketched data.

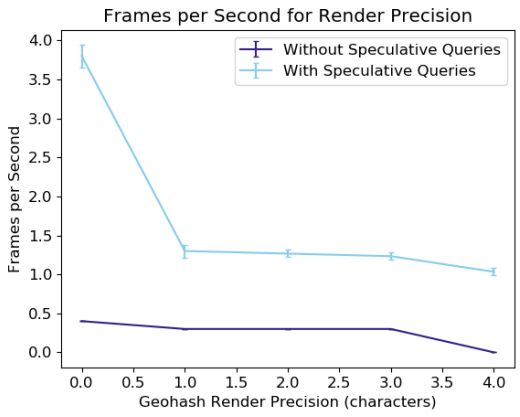

Figure 9: The number of frames in an animation that Aperture can display each second depending on the render precision. Rendering precision has a relatively small effect on animation speed, except at a precision of zero, in which Aperture is rendering a single polygon.

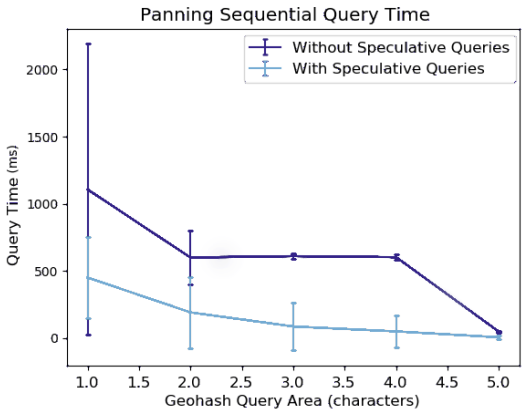

Figure 10: The speed with which Aperture can load queried geohashes during a panning operation, with and without speculative queries. The speculative queries provide a clear benefit, often cutting the query time by more than half.

which the user starts viewing data at a given point in time, then repeatedly advances the time by a fixed amount. Here, Aperture's performance is measured in the number of different times that can be viewed per second. This translates directly to the smoothness of the animation and the amount of data that can be displayed in a given time. Aperture's animation speed, with and without speculative queries, is shown in figure 8 and figure 9.

The second task used to measure Aperture is a panning task, in which a user begins at a randomly chosen <latitude, longitude> and pans across the map to another randomly chosen <latitude, longitude> pair. The direction of the pan follows the shortest path between the two points at a constant speed and may diagonally cut across several geohashes, not necessarily following a single row or column.

In figure 10 it can be seen that speculative queries provide a consistent performance boost, except at a very small query area. This is due to the speed of queries with small areas rather than a defect in the speculative queries.

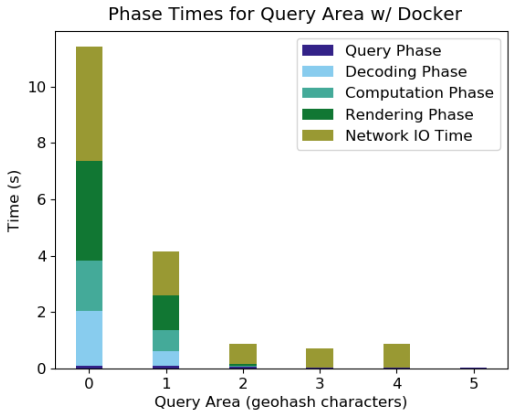

Figure 11: Aperture's phase times when run from a Docker container in a Kubernetes cluster. The time for each phase is approximately doubled, with the exception of the query phase.

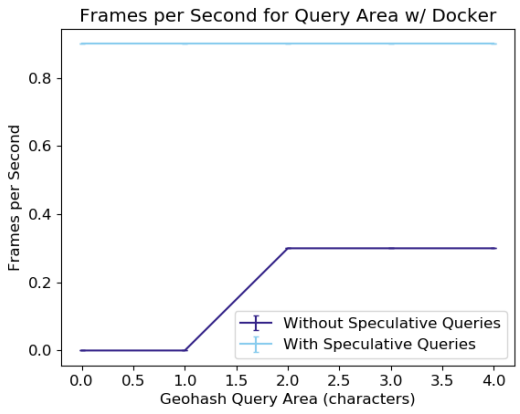

Figure 12: The number of frames in an animation that Aperture can display each second when run in a Docker container in a Kubernetes cluster. Animations load at 40% the speed they do without Docker.

4.6 Docker Performance Profile

We ran our Docker containers in a Kubernetes[11] cluster with horizontal auto-scaling enabled, exposed via a NodePort service. Kubernetes initially places containers on the machines it measures as least busy while auto-scaling allows it to replicate highly-utilized containers to keep resource consumption across the cluster near a target value. In theory, Kubernetes provides high availability and reduces latency by efficiently apportioning the workload.

In figure 11 it can be seen that running Aperture requires approximately twice as much time to perform queries under these conditions, likely due to the network and CPU overheads involved in running a container and communicating through a service. This difference is insignificant for query areas of more than a single character, but for large query areas it becomes quite substantial. In figure 12 the total performance dropped by more than 50%. This is likely due to the fact that when containers are killed or moved periodically they lose their cache, which is critical for increasing animation speed. The performance of statistical queries through Docker was unchanged.

Despite the loss in performance, Docker still provides tangible advantages, especially when dealing with hotspots. Hotspots occur when a specific temporal location recieves an enormously disproportionate number of queries, such as when tracking localized weather phenomena e.g. a hurricane. In such cases containers would be invaluable for transparently scaling the most critical spatiotemporal scopes, allowing faster, predictable response times during visualizations under high loads.

5. Conclusions and Future Work

In this paper, we have described our methodology for effective visualization of spatiotemporal phenomena. Our methodology makes novel use of sketches to accomplish this goal.

RQ-1: To cope with data volumes and the speed differential of the memory hierarchy we leverage sketches. Sketches extract information from the data, are data-format agnostic, and are three orders of magnitude more compact than the raw data. We leverage compactness of the sketches by pinning

them in memory, where access to them is not subject to the high latencies and low throughputs of the disk I/O subsystem. Leveraging sketches also significantly reduces the amount of data that needs to transferred from the server-side.

RQ-2: Visualization involves several processing tasks, implicit and explicit, that must be orchestrated in concert to ensure effective visualization. Accomplishing this involves apportioning of these loads at the client and server-side. Minimizing the amount of network I/O triggered by visualization tasks, especially those in the critical path, are key to ensuring responsiveness. Effective sifting of observations through query construction based on the view-port is also necessary to reduce workloads and I/O.

RQ-3: To support effective multi-linked views with brushing-and-linking, we leverage the structural properties of the sketch. Since the sketch is organized as a tree, we traverse the sketch to perform aggregations that are needed in some views. Connecting Choropleth views with other views involves identifying the selected polygons, dynamically constructing metadata queries over the sketch, and then performing aggregation operations.

RQ-4: Interactivity during visualization is preserved by effectively apportioning workloads between the client and server-side. Furthermore, since data access patterns often exhibit temporal locality, they benefit from our caching at the server-side and also the buffering that we perform at the server side. Our use of sketches facilitates attenuated disk and network I/O. Because our sketches are distributed, server-side processing is distributed as well. This facilitates fast completion times.

RQ-5: As part of future work, we will explore inter-operation with the two of the dominant spatial analyses ecosystems: ESRI's ArcGIS and Google Earth Engine. Our efforts will focus on incorporating support for the sketch as a data type within these APIs and leveraging visual analytics capabilities provided within these ecosystems.

Acknowledgements

This research was supported by funding from the US National Science Foundation [OAC-1931363] and the US Department of Homeland Security [D15PC00279].

References

[1] John A Hoxmeier and Chris Dicesare. 2000. System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications. *Proceedings of the Association of Information Systems Americas Conference* (01 2000).

[2] Shamim Bhuiyan, Michael Zheludkov, and Timur Isachenko. 2017. *High Performance In-memory Computing with Apache Ignite*. Lulu.com.

[3] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec. 2011), 2301–2309. DOI: http://dx.doi.org/10.1109/TVCG.2011.185

[4] T. Buddhika, M. Malensek, S. L. Pallickara, and S. Pallickara. 2017. Synopsis: A Distributed Sketch over Voluminous Spatiotemporal Observational Streams. *IEEE Transactions on Knowledge and Data Engineering* 29, 11 (Nov 2017), 2552–2566. DOI: http://dx.doi.org/10.1109/TKDE.2017.2734661

[5] Josiah L. Carlson. 2013. *Redis in Action*. Manning Publications Co., Greenwich, CT, USA.

[6] Kristina Chodorow. 2013. *MongoDB: The Definitive Guide*. O'Reilly Media, Inc.

[7] Paul Crickard. 2014. *Leaflet.Js Essentials*. Packt Publishing.

[8] B. Delaunay. 1934. Sur la sphere vide. A la memoire de Georges Voronoi. In *Bulletin de l'Academie des Sciences de l'URSS. Classe des sciences mathematiques et na*. 793–800.

[9] Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. 2013. Sketch-based Geometric Monitoring of Distributed Stream Queries. *Proc. VLDB Endow.* 6, 10 (Aug. 2013), 937–948. DOI: http://dx.doi.org/10.14778/2536206.2536220

[10] Marios Hadjieleftheriou, John W. Byers, and George Kollios. 2005. *Robust sketching and aggregation of distributed data streams*. Technical Report.

[11] Kelsey Hightower, Brendan Burns, and Joe Beda. 2017. *Kubernetes: Up and Running Dive into the Future of Infrastructure* (1st ed.). O'Reilly Media, Inc.

[12] Q. Huang and P. P. C. Lee. 2014. LD-Sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 1420–1428. DOI: http://dx.doi.org/10.1109/INFOCOM.2014.6848076

[13] Jiawei Jiang, Fangcheng Fu, Tong Yang, and Bin Cui. 2018. SketchML: Accelerating Distributed Machine Learning with Data Sketches. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, New York, NY, USA, 1269–1284. DOI: http://dx.doi.org/10.1145/3183713.3196894

[14] J. Koontz, M. Malensek, and S. Pallickara. 2014. GeoLens: Enabling Interactive Visual Analytics over Large-Scale, Multidimensional Geospatial Datasets. In *2014 IEEE/ACM International Symposium on Big Data Computing*. 35–44. DOI: http://dx.doi.org/10.1109/BDC.2014.12

[15] L. Lins, J. T. Klosowski, and C. Scheidegger. 2013. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (Dec 2013), 2456–2465. DOI: http://dx.doi.org/10.1109/TVCG.2013.179

[16] M. Malensek, S. L. Pallickara, and S. Pallickara. 2011. Galileo: A Framework for Distributed Storage of High-Throughput Data Streams. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*. 17–24. DOI: http://dx.doi.org/10.1109/UCC.2011.13

[17] MapBox 2019. MapBox. https://www.mapbox.com/. (2019).

[18] Dirk Merkel. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014, 239, Article 2 (March 2014). http://dl.acm.org/citation.cfm?id=2600239.2600241

[19] Saptashwa Mitra, Paahuni Khandelwal, Shrideep Pallickara, and Sangmi Lee Pallickara. 2019. STASH: Fast Hierarchical Aggregation Queries for Effective Visual Spatiotemporal Explorations. In *(To appear) IEEE International Conference on Cluster Computing (CLUSTER)*. Albuquerque, NM, USA.

[20] Balearic Island Coastal Observing and Forecasting System. 2019. Leaflet.TimeDimension. (Aug. 2019). https://github.com/socib/Leaflet.TimeDimension

[21] C. Stolte, D. Tang, and P. Hanrahan. 2003. Multiscale visualization using data cubes. *IEEE Transactions on Visualization and Computer Graphics* 9, 2 (April 2003), 176–187. DOI: http://dx.doi.org/10.1109/TVCG.2003.1196005

[22] Tableau 2019. https://www.tableau.com/. (2019).