THESIS

APERTURE: A SYSTEM FOR INTERACTIVE VISUALIZATION OF VOLUMINOUS GEOSPATIAL DATA

Submitted by

Kevin Bruhwiler

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2020

Master's Committee:

Advisor: Shrideep Pallickara
Co-Advisor: Sangmi Lee Pallickara

Sudipto Ghosh
Venkatachalam Chandrasekaran

Copyright by Kevin Conrad Bruhwiler 2020

All Rights Reserved

ABSTRACT

APERTURE: A SYSTEM FOR INTERACTIVE VISUALIZATION OF VOLUMINOUS GEOSPATIAL DATA

The growth in observational data volumes over the past decade has occurred alongside a need to make sense of the phenomena that underpin them. Visualization is a key component of the data wrangling process that precedes the analyses that informs these insights. The crux of this study is interactive visualizations of spatiotemporal phenomena from voluminous datasets. Spatiotemporal visualizations of voluminous datasets introduce challenges relating to interactivity, overlaying multiple datasets and dynamic feature selection, resource capacity constraints, and scaling. Our methodology to address these challenges relies on a novel mix of algorithms and systems innovations working in concert to ensure effective apportioning and amortization of workloads and enables interactivity during visualizations. In particular our research prototype, Aperture, leverages sketching algorithms, effective query predicate generation and evaluation, avoids performance hotspots, harnesses coprocessors for hardware acceleration, and convolutional neural network based encoders to render visualizations while preserving responsiveness and interactivity. Finally, we also explore issues in effective containerization to support visualization workloads. We also report on several empirical benchmarks that profile and demonstrate the suitability of our methodology to preserve interactivity while utilizing resources effectively to scale.

ACKNOWLEDGEMENTS

I would like to thank Shrideep and Sangmi Pallickara for their support throughout my time at CSU, both during my undergraduate and graduate programs, and for believing in my potential as an academic and giving me opportunities when few others would. I would also like to thank Thilina Buddhika for his significant contributions to this work, both direct and indirect – Aperture depends heavily on systems he developed. Let me also express my appreciation for the Systems and Network Administration in CSU's Computer Science Department for their extraordinary patience with my many questions and my perpetually growing list of software requirements. Finally, thank you to my family for the reliable emotional and financial support, and apologies to my friends, who tolerated multiple rants about JavaScipt and geospatial projection with good-natured humor.

TABLE OF CONTENTS

Chapter 1

Introduction

The proliferation of observational devices, improvements in the resolution and frequency at which measurements have been made, and falling costs for data storage have all contributed to an increase in data volumes. These data volumes hold the potential to unlock insights via data analytics. A key intermediate step in the data wrangling process that precedes the analyses is visualization. Visualization allows scientists to quickly assess broad patterns in the data.

This study focuses on spatiotemporal data where data is tagged with spatial information representing the location being observed and the timestamp reflecting when these measurements were made. These spatiotemporal data represent a substantial portion of the cumulative data volumes. Such data occurs in social media, observational and telemetry settings, transportation networks, simulations, and commerce among others. Spatiotemporal visualization allows us to identify spatial extents, temporal segments, or some combination thereof — also referred to as the spatiotemporal scope of interest. Identifying such spatiotemporal scopes allows practitioners to target their modeling efforts more precisely.

To maximize interoperability across different platforms we use the browser as the primary gateway for visualizations. Unlike traditional client-server interactions our visualizations entail interactions with a server-side that encompasses a distributed collection of machines. The distributed server-side is responsible for managing requests from multiple, diverse clients concurrently. [1, 2] [1]

[1] The majority of this work is the direct result of the following two papers:

Kevin Bruhwiler and Shrideep Pallickara. 2019. Aperture: Fast Visualizations Over Spatiotemporal Datasets. In Proceedings of IEEE/ACM 12th International Conference on Utility and Cloud Computing, Auckland, New Zealand, December 2–5, 2019 (UCC '19),10 pages

Kevin Bruhwiler, Thilina Buddhika, Shrideep Pallickara, and Sangmi Lee Pallickara. 2020. Iris: Amortized, Resource Efficient Visualizations of Voluminous Spatiotemporal Datasets. In Proceedings of the IEEE/ACM 7th International Conference on Big Data Computing, Applications and Technologies (BDCAT'20) **(Accepted)**

1.1 Challenges

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

- **Selective Overlays**: Often patterns are easier to visualize when the analyses is supplemented by auxiliary datasets. In particular, this involves allowing overlaying of features (also referred to as independent variables) from diverse datasets.
- **Data Volumes**: Visualization involves a mix of disk accesses, network transfers, computation of visual artifacts, and memory residency. Data volumes exacerbate these aforementioned challenges by exceeding resource utilization thresholds or capacities.
- **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.
- **Interactivity**: To be useful, visualizations must be interactive, allowing practitioners to maintain their chain-of-thought and enabling visualizations to inform their explorations. The visualizations must preserve this interactivity during key explorative operations such as drill-down, roll-up, and panning across spatiotemporal scopes.
- **Scale**: Visualization systems must scale with increases in the number of clients. This entails minimizing interactions between the client and the server. Functionality must be effectively apportioned so that the client-side takes on a large portion of the load.

1.2 Research Questions

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

- How can we effectively harness available resources and amortize workloads (CPU, memory, disk, and network I/O) to cope with data volumes and the speed differential of the memory hierarchy? Effective amortization of workloads guides this investigation.

- How can we leverage perceptual characteristics/limits of visualization to improve the timeliness of rendering operations? This involves rendering coarser visual artifacts that are then incrementally refined.
- How can we leverage learning during visualization? There are two aspects to this. The first aspect involves predicting spatiotemporal paths during visualization. The second aspect involves leveraging deep learning to render the phenomena, rather than computing the polygons comprising the visualization.

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 leverage the Synopsis sketching algorithm [3]. Synopsis is a single-pass sketching algorithm which produces statistical sketches from the data. Once the sketch is constructed it is the sketch and not the original data that is consulted during visualization operations. The sketch is a space-efficient data structure that serves as a surrogate for the voluminous data that it sketches. In particular, the Synopsis algorithm extracts information from the data and tracks distributional characteristics of the data, summary statistics, and cross-feature covariances. We have extended the basic Synopsis algorithm to facilitate efficiency gains from a compaction perspective. The sketch is broken up into a collection of space-efficient *strands*. Each strand encapsulates information for a particular spatiotemporal scope. The strands are amenable to aggregation and two strands can be combined to produce a single strand for the larger spatiotemporal scope.

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

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 center them within the queried bounds.

We support two different configurations: Server-side and client-side rendering. In server-side rendering the Delaunay triangulation [4] 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.

Our server-side is designed to enable high query throughputs while maintaining a lower query latency. Storage nodes are arranged as a distributed hash table (DHT) to provide better scalability and fault-tolerance. Our data dispersion criterion ensures both near-uniform distribution of data across the DHT while preserving temporal locality, which reduces disk IO during query evaluation. Data is indexed both spatially and temporally for faster lookups. Further, we leverage multi-core architectures, disk caching, and fast data serialization schemes effectively for efficient query processing.

In client-side rendering, preserving responsiveness is the priority. Preserving responsiveness is a key requirement during visualizations. Traditional batched visualizations are easy to implement and reason about. However, batched visualizations have inefficiencies that stem from synchroniza-

tion barriers that exist between each phase (query, retrieval, and rendering) of the visualization. Batched visualizations introduce lag because each phase cannot start before the preceding one fully completes, contributing to prolonged wait times. Furthermore, batched visualizations are computationally expensive and require all data to be available before performing computations. As a result, they suffer resource spikes that induce failures.

In Aperture, rather than retrieve results at all once, the results are streamed from the server to the client. The rendering computations are aligned with this streaming. In particular, the query retrievals and rendering operations are continually interleaved, relieving resource spikes. Furthermore, because our incremental rendering operations amortize the workloads associated with rendering operations, responsiveness is preserved as well. This process is combined with quantization (where we reduce the precision of features) when computing the visual artifacts and rendering. These are incrementally refined as additional data become available.

We also leverage client-side deep learning for visualizations, which involves two key steps: encoding and mapping. We use an encoder to first learn a compact, lower-dimensional encoding of a given spatiotemporal range that can then be used to create a visualization with high fidelity. The second step involves a mapping phase where the encoded feature space is combined with a user query to produce the visualization. This approach replaces piece-wise calculation of visual artifacts within an image with a series of matrix/tensor-based operations that are highly amenable to acceleration on coprocessors at the client side. The computationally expensive operation of training the encodings is performed on the server-side away from the critical path of client interactions. Only trained models are installed on the client side. Besides reduction in memory and network overheads, another benefit of our methodology is that, because geohashes are included as a feature vector, we are able to reconcile spatial heterogeneity.

Additionally, we 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 are kept at a minimum.

Our methodology makes effective use of resources at the client-side. In particular, we leverage GPU libraries both during rendering operations and also during inferences performed by our deep CNN (convolutional neural network) based encoder. Our systems benchmarks demonstrates the suitability of our methodology to leverage hardware acceleration at the client side. The experiments were performed with well-known voluminous, multidimensional datasets.

1.4 Contributions

Our methodology facilitates visualization of voluminous spatiotemporal datasets at scale. In particular, our contributions include:

1. Effective distribution of visualization workloads to reduce strain on the server side, minimize network communications, and alleviate memory pressure.

2. We leverage co-processors (GPU) to make effective use of client-side capabilities both during calculation of visual artifacts and deep learning operations at the client side (inferences) and server-side (model training).

3. 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.

4. Effective amortization of workloads by combining streaming of results with incremental refinement of the visualization to balance responsiveness. Does not introduce hotspots because the computations are amortized.

5. A deep learning-based framework to generate visualizations on the fly. Our methodology performs a novel mapping from the feature space to the latent encoding space to produce these visualizations.

6. We can support animations of voluminous spatiotemporal phenomena at two frames/second.

7. 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.

Chapter 2

Related Work

2.1 Visualization Tools

There are many tools for data visualization designed to function with a variety of data and storage systems. Data Cubes [5] 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 [6] is a powerful data visualization tool specializing in business intelligence. Its usefulness derives from its ability to handle data from many different sources and 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 [7] 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 and 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, either performs individual queries on a single machine, allowing it to produce much higher-resolution visualizations at a slower pace (compensated for with speculative queries and a server-client caching scheme), or progressive client-side rendering which enables much faster visualizations (see section 3.2.1).

2.2 Incremental Visualization

Incremental visualization tools depend on data streams, in which the data being visualization arrives continuously over time, rather than all at once. Creating visualizations with data streams has been explored in previous works with specialized techniques for time-series data [8], creating visualizations in parallel using data streamed from simulations [9], and using kernel density estimation functions with GPU support [10]. Aperture synthesizes several of these ideas, including leveraging the horizontal scalability of distributed computing clusters to maximize bandwidth and using client-side graphics processing units (GPUs) to improve rendering times. Aperture also takes advantage of advances in computing technology, including efficient serialization with protocol buffers [11], browser-GPU inter-operation with WebGL [12], and HTML canvas [13], to achieve improved performance.

Several methods of incremental visualization generation are able to provide visual analysis of datasets that are prohibitively slow or expensive to visualize in full [14–18]. However, they all depend on batched visualization (in which data is collected client-side and periodically re-rendered) and suffer from one of its major drawbacks: a certain amount of data must be retrieved before visualizations are recomputed and rendered, limiting interactivity. In Aperture we present an efficient method for incrementally refining visualizations as individual data points arrive, preserving both interactivity and resolution.

2.3 Data Sketching

Data sketching defines a set of techniques which are used to build a probabilistic approximation of a dataset, 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 in conjunction with machine learning algorithms [19]. Aperture makes use of a distributed sketching algorithm called Synopsis (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 [20], anomaly detection [21], and threshold monitoring tasks [22].

Sketch-based storage of spatiotemporal data has been explored before [3,23,24]. The Synopsis sketching algorithm has been used previously to build a spatiotemporal data store. Its in-memory storage model is in contrast to its on-disk storage model. On-disk storage improves the scalability of the system due to the capacity differential of physical memory and disk space available in commodity hardware clusters. Aperture relies on various optimization techniques such as parallel disk IO, parallel query execution, and disk caches to counter the IO speed differential between spinning disks and memory. Aggregate RB trees (aRB trees) [23] are used to answer spatiotemporal count queries. Aperture stores spatial regions in R-trees as bounding rectangles. Each bounding rectangle points to a B-tree where historical aggregates of the feature values are stored. Tao et al. [24] extends aRB trees with the FM sketching algorithm [25] to answer distinct count queries. However, unlike Aperture, these systems are designed to work with a single feature stream and support only count based queries.

2.4 Data Storage

Aperture relies on Synopsis to store geospatial data and process spatiotemporal queries, but 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. A survey of data management challenges in cloud-scale settings can be found in [26]; our methodology targets addressing data volumes by leveraging sketches of the data.

MongoDB [27] 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 [28] 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. The Galileo system incorporates support for approximate [29], analytic [30], and polygon-constrained [31] queries over spatiotemporal datasets. Converting data stored by Galileo into a Synopsis sketch is trivial, making Galileo another good choice for an underlying database.

Stash [32] 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.

Efforts have also been made to optimize queries over stored geospatial data. Geometry constrained query evaluations have been explored in the context of disk resident data [33]. This work targets query evaluations over sketches.

Aperture owes some 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 [34] 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 [35], 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, but this would require the development of a new system which can aggregate spatiotemporal data in a manner easily understood by Ignite or Redis.

Nanocubes [36] 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 that they can use to construct visualizations to what can be stored in 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 [37] in **section 3.5**.

Chapter 3

Methodology

3.1 Sketches

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. The Synopsis systems relies on Neptune [38, 39] for the ingestion and routing of spatiotemporal streams. 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 3.1) to partition and agglomerate spatiotemporal observations.

The Synopsis sketch is organized as a forest of trees (called strands, see section 3.1.2), which maintain 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 organizational 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.

3.1.1 Geohashes

The geohash algorithm (Figure 3.1) 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 3.1 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

Table 3.1: Geographical extent of different geohash precisions, measured in number of characters, located near the equator.

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	Neighborhood

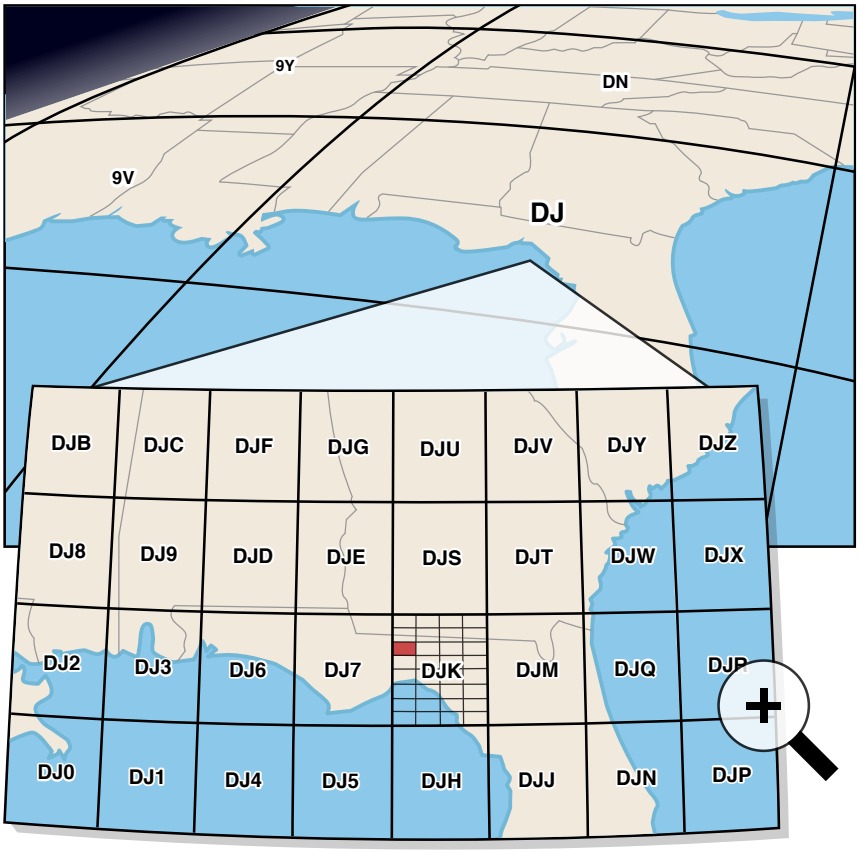

Figure 3.1: The geohash algorithm. Adding characters to the right side of the string reduces the area covered by the geohash and represents a more precise location.

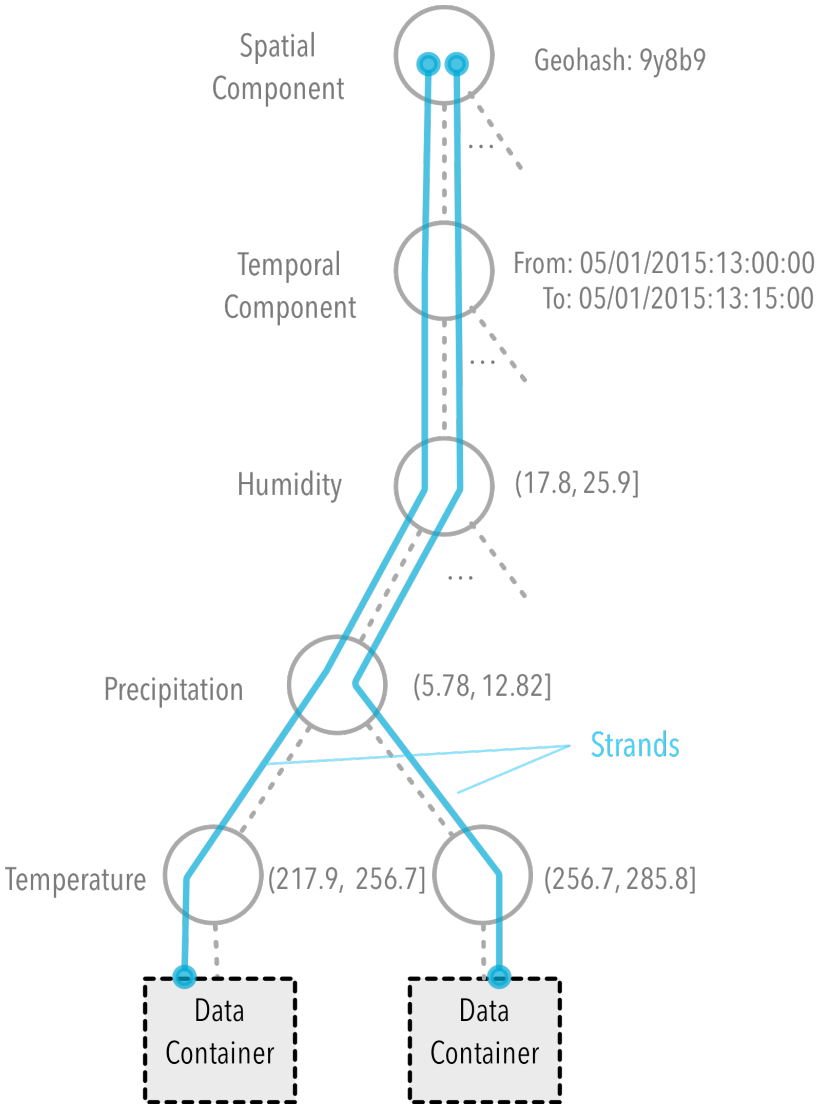

Figure 3.2: An example of strand construction using Synopsis sketching algorithm. Spatio-temporal components and individual features are represented in coarser-grained resolutions to reduce the storage footprints. In this example, the subtree rooted at the vertex for temporal component summarizes all the records occurring in an area of $4.9 \times 4.9km^2$ represented by the geohash `9y8b9` for the 15 minute time interval starting at `13:00:00` on `05/01/2015`.

3.1.2 Strands

The design premise of the Synopsis sketching algorithm [3] is based on representing individual feature values at a coarser-grained resolution while preserving inter-feature relationships in order to reduce data size. This process is called *discretization*. Discretization represents each feature value as a record using a bin based on a predetermined bin configuration. A bin configuration

comprises a set of non-overlapping intervals that collectively construct the range for a particular feature.

For instance, suppose the bin configuration for surface temperature (in Kelvin) is `[217.9, 256.7), [256.7, 285.8), [285.8, 305.2), [305.2, 334.3)`. If the surface temperature is `220.31` of a particular record, then it falls within the range of the first bin, `[217.9, 256.7)`, therefore represented using the first bin. Similarly, if the temperature slightly increases to `220.32` in the next record, the updated feature value still gets mapped to the same bin.

In real settings, bin configurations are more dense — most bin configurations contain around 30-50 bins. Bin configurations are calculated using a kernel density estimation based method such that the overall error due to discretization is maintained below a given threshold. We use the normalized root mean square error (NRMSE) of 2.5% as our error threshold for the benchmarks. Further, our storage framework supports the evolution of bin configurations over time to capture concept drifts (e.g., different bin configurations for Winter and Summer seasons for a given region).

Spatial and temporal components of observation are also represented at coarser resolutions using geohashes. The geohash algorithm [40] produces a deterministic mapping of 2-dimensional spatial extents into a 1-dimensional string; the length of the geohash string is inversely proportional to the size of the spatial extent – the longer the geohash, the smaller the spatial extent that it represents. We use a prefix of the geohash to represent the spatial component — this maps multiple observations that occur within spatial proximity onto the same geohash prefix. Similarly, temporal components are also mapped to coarser-grained time intervals.

The discretized records are then collated into a tree-like data structure, as shown in Figure 3.2. Each tree path ending with a unique leaf node is called a *strand*. Coarser-grained resolutions improve the compaction of the dataset by mapping multiple records into fewer strands. Records that occur within a particular spatial and temporal proximity will be likely to be mapped into a single strand after being discretized. At leaf nodes, a set of online statistics are maintained to summarize all records that are represented using that particular strand. We use Welford's method [41] to maintain the number of observations, the running mean, the sum of squares of differences from

the current mean, and the sum of cross products between features. These statistics are useful for calculating descriptive statistics as well as inter-feature relationships. We use strands as the unit of data storage in our backend system.

3.1.3 Aperture without Synopsis

Visualization is underpinned by queries over the observational space, and the efficiency of these evaluations 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 queries while sidestepping Synopsis and reading data directly from the hard drive. The comparative performance is depicted in Figure 3.3. 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 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 3.3 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 pre-load data that may be viewed in the future. 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

Users interact with Aperture via a web page which displays the data over a pannable and zoomable choropleth map of the world (Figure 3.4). 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

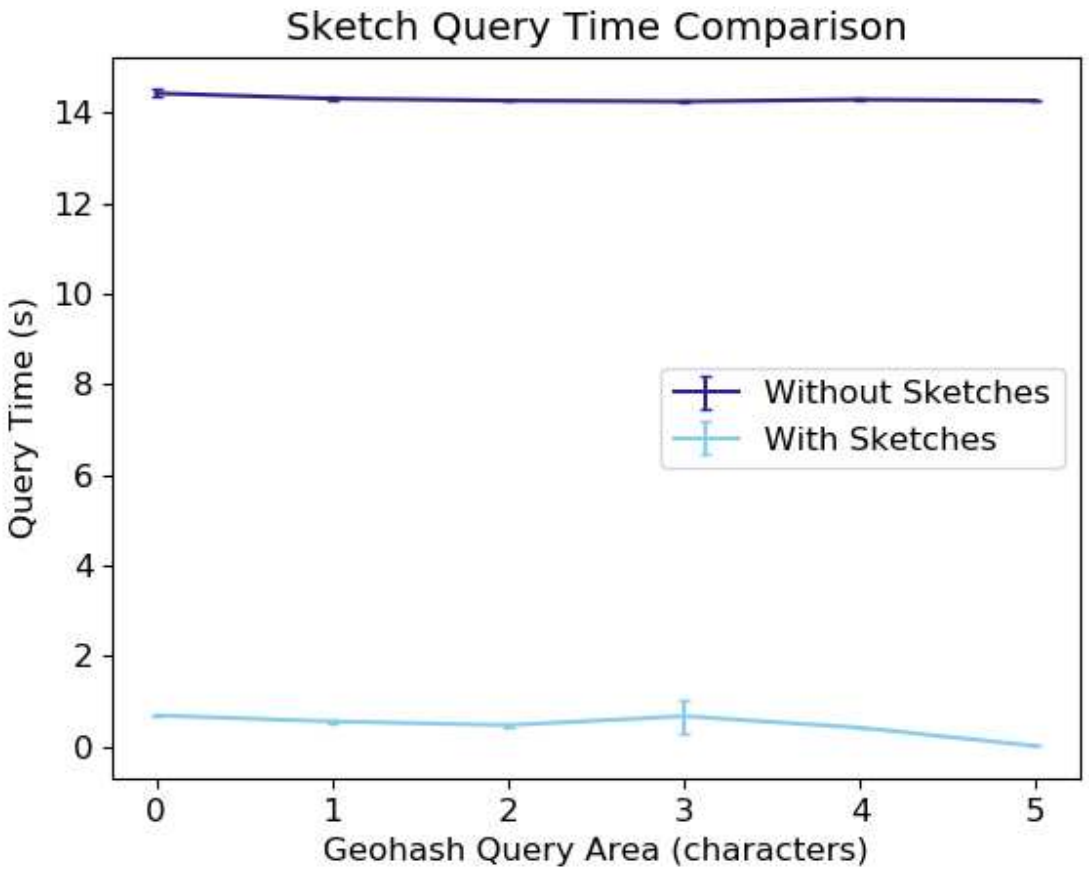

Figure 3.3: A comparison of the time Aperture takes to complete a query with and without Synopsis. Performing queries on sketches is more than 14 times faster than queries on un-sketched data.

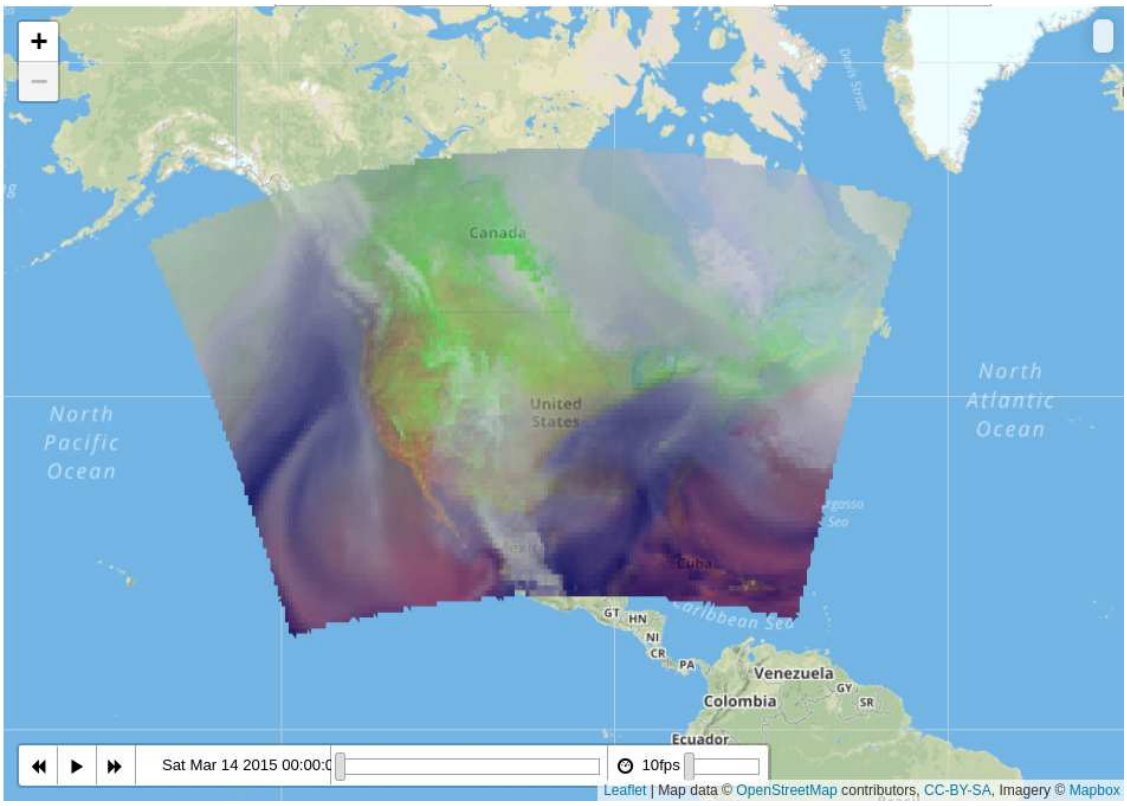

Figure 3.4: Aperture's display. The map consists of tessellated polygons, each colored to represent measurements within their area. Temperature (red-green), humidity (white), and precipitable water (purple) are shown.

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 leverages Leaflet to render spatiotemporal phenomena. Leaflet [42] 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.

Aperture utilizes Leaflet for two purposes. Firstly, it is used in conjunction with MapBox [43] 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 utilizes Leaflet Time Dimension layers [44], developed by the Balearic Island Coastal Observing and Forecasting System (SOCIB), to assist in displaying animations over temporal ranges.

We have tested two different configurations for Aperture. The first depends on the server to raster the dataset into an image, which is sent to the client to display. This method is advantageous in that it enables Aperture to use the server's resources, which are typically much greater than the client's, to construct a high-quality visualization at the cost of higher latency and lower interactivity. The second configuration depends on the client to render individual data points as they arrive, resulting in dramatically lower latency at the cost of some fidelity. This trade-off is explored further in chapter 4.

3.2.1 Containerized Server-Side Rendering

In the server-side rendering configuration, 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. 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.

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 [37] 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 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. This measure also fails 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.

3.2.2 Incremental Client-Side Rendering

Incremental client-side rendering, in which visualizations are computed entirely on the client, depends on the Synopsis server replying to queries with a stream of data strands. The cumulative network footprint of the strands matched with a particular query can vary from a few Kilobytes to hundreds of Gigabytes depending on the spatiotemporal scope collectively constructed by the predicates. Instead of returning all matching strands at once, we stream matching strands to the client as soon as they are retrieved by the worker threads.

Streaming results and handling them incrementally is beneficial for visualizations because: (1) the visualization is updated immediately after any user interaction, dramatically improving the perceived responsiveness of the system, and (2) expensive computations required to render the visualization are amortized, minimizing interference with other processes on the client machine and reducing the odds of the client suffering load-related failures. Further, a streaming query model reduces the strain on the memory of the server by shortening the memory retention period of the results, therefore improving the overall query throughput of the system.

We designed our query API to be language agnostic by using gRPC/Protocol Buffers — allowing clients implemented using numerous supported programming languages to interact with our storage system. We group multiple strands into a single message to improve the network bandwidth utilization.

The Synopsis query API is exposed through an array of proxy services acting as the gateway to the storage system. Once a query is received, the proxy server forwards it to the DHT, triggering a set of response streams originating from individual DHT nodes. These streams are then merged into a single stream at the proxy server and funneled back to the client. Proxy nodes can often

become bottlenecks due to saturated resources such as network bandwidth and CPU. To counter this issue, new proxies can be added on-demand without disrupting the ongoing queries. The stateless runtime design of proxy servers enables the seamless horizontal scaling of the proxy array.

Data arrives at the client as a stream of Synopsis strands, serialized as protocol buffers. The client is responsible for deserializing them, converting the geohash location to a <*latitude, longitude*> point, calculating the color of the associated polygon from the deserialized features, and passing this information to the HTML canvas. The client performs these operations on each data strand asynchronously to allow it to cope dynamically with changes in the rate of data arrival.

Sketches are rendered on an HTML canvas in real-time as strands arrive at the client, allowing users to visualize data points the moment that they are available. The canvas element is a low level model in HTML used for rendering bitmaps and 2D graphics. It can optionally make use of WebGL which enables 3D rendering and hardware acceleration via a GPU.

Due to the high rate of strand arrival it is critical that the canvas can render each strand quickly. Because canvas elements can be updated incrementally, where only the modified region of the canvas is recomputed rather than the entirety, it is well-suited to rendering streamed data. Additionally, we ensure that the canvas is not performing expensive anti-aliasing operations by rounding the location of each polygon so that its bounds align evenly with the canvas' pixels.

On the canvas each strand is represented as a square polygon of a dynamic size, the color of which is determined by the strand's features. The size of each polygon is computed using one of several decay algorithms (see Section 4.3): the result is that the first few polygons are extremely large, taking up much of the visualization area, while the size of later polygons is rapidly reduced down to a fixed minimum. This has the effect of immediately providing users with a coarse-grained visualization that becomes increasingly refined as more data arrives. The incremental rendering process is visualized in Figure 4.8.

3.2.3 Rendering with Deep Neural Networks

We experiment with using deep convolutional networks to quickly generate visualizations on the client. We explore the ways in which this can improve the performance of Aperture in three major respects:

1. Reducing network traffic by eliminating the need to re-run queries server-side when users change the spatial, temporal, or feature predicates of the visualization within a predefined temporal range.

2. Improving the responsiveness of data visualizations by eliminating the need to wait for data to be streamed from the server.

3. Reducing computational load at the client by replacing CPU-intensive deserialization operations with matrix multiplications highly amenable to co-processor acceleration.

We also investigate the uncertainty that comes from using generated visualizations and the potentially deleterious effects on the final resolution.

Training

The training dataset was created and curated by generating high quality visualizations for a large number of user queries, then saving the query and the visualization along with all of the data strands for that particular temporal scope. This resulted in a set of *(input, target)* pairs where the input consisted of both a user query and all of the strands from the encompassing temporal scope that could potentially be used to generate the associated visualization. The strands are grouped into a matrix that preserves the geospatial relations between the data points, allowing the use of convolutional layers.

The network was trained in a distributed computing cluster using PyTorch [45], both to make use of additional cores and help cope with the relatively slow process of reading and evaluating the (input,target) pairs. The error was calculated by averaging the binary cross-entropy loss [46] (BCE) between every pixel in the generated image and the target visualization. Gradients were computed

to minimize the BCE loss and the weights of the network were updated using the Adam [47] stochastic optimizer.

Network Structure

The network is structured as a typical image-to-image translation model, using convolutional layers to transform one image to another. The user query is encoded as an additional channel and appended to the matrix of input strands.

Due to the limited types of matrix operations that can be executed on the client the generator is required to have an unusual structure. Rather than using the traditional up-sampling or deconvolutional layers to generate images common in GANs and VAEs, we use a series of convolutional layers with a low number of filters that do not change the size of the image. This allows us to transform data into a generated rendering within the contraints of web-based deep learning while still taking advantage of the spatial nature of convolutions.

Mapping Phenomenon

To run the trained network on the client-side we first convert the trained PyTorch model to Onnx [48], a protocol buffer based machine learning model representation format. We use Onnxjs, which has WebGL support, to execute the model on the browser with co-processor support. Onnxjs was chosen over Tensorflowjs [49] due to higher compatibility with different libraries and improved speed during model evaluation [50].

3.3 Query Generation and Refinement

In Aperture, queries are intelligently generated based on the region of the map currently being viewed and the degree to which the view is zoomed in or out. The period of time being queried is set by the user, and constrained by the frequency of the data.

3.3.1 Spatial Predicates

The spatial predicates for each query are determined by a geohash-based tree search. The smallest bounding geohash is computed by taking the longest matching geohash of the top-right and bottom-left corners of the visualization area. For example, if the top-right geohash is "9zgvkpbsf" and the bottom-left is "9zjs22e3h", the shortest matching prefix would be "9z". Sub-geohashes are then searched recursively: the bounding box of each geohash is compared to the bounding box of the visualization area to determine if there is any overlap. If the boxes do overlap, that geohash is added to the search set and the process continues down to a predefined precision. Due to overheads in querying many geohashes simultaneously, the target precision is determined based on the current zoom level of the visualization so that there will never be too many geohashes in a single query.

3.3.2 Temporal Predicates

The temporal range of a query is determined based on two factors: the frequency of the sketched data and a user-selected time. All sketched data is bucketed into a configurable temporal bracket (see Section 3.4). Each query specifies a single temporal bracket, although querying multiple brackets is possible. The bracket is determined via a Time-Dimension UI [44] that allows users to query a specific time or play an animation across a range of times by performing many sequential queries.

3.3.3 Speculative Queries

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.

While Aperture is capable of generating speculative queries in both server-side and client-side rendering configurations, during client-side rendering Aperture makes optimal use of client resources. This means that simultaneously computing multiple visualizations reduces client responsiveness linearly with respect to the number of visualizations being computed (computing two at once means both will take twice as long, three will take thrice as long etc...). Consequently, speculative queries are only performed while no other visualization is being actively generated. In this case, precomputed visualizations are stored in an LRU cache on the client and rendered as needed.

3.4 Query Evaluation

Our storage subsystem is optimized for queries (read traffic). We employ an array of optimizations such as uniform data dispersion, data locality, indexing, parallel query execution, parallel disk I/O, and caching for efficient data retrieval. Figure 3.5 depicts the high level architecture of our query model.

We arrange the storage nodes as a DHT with a consistent hashing-based data dispersion scheme based on a key generated by combining the spatial attribute and the temporal attribute of a strand. DHTs provide better load balancing, incremental scalability, ability to work with heterogeneous commodity hardware, and fault tolerance. We use the geohash of a strand combined with a coarser-

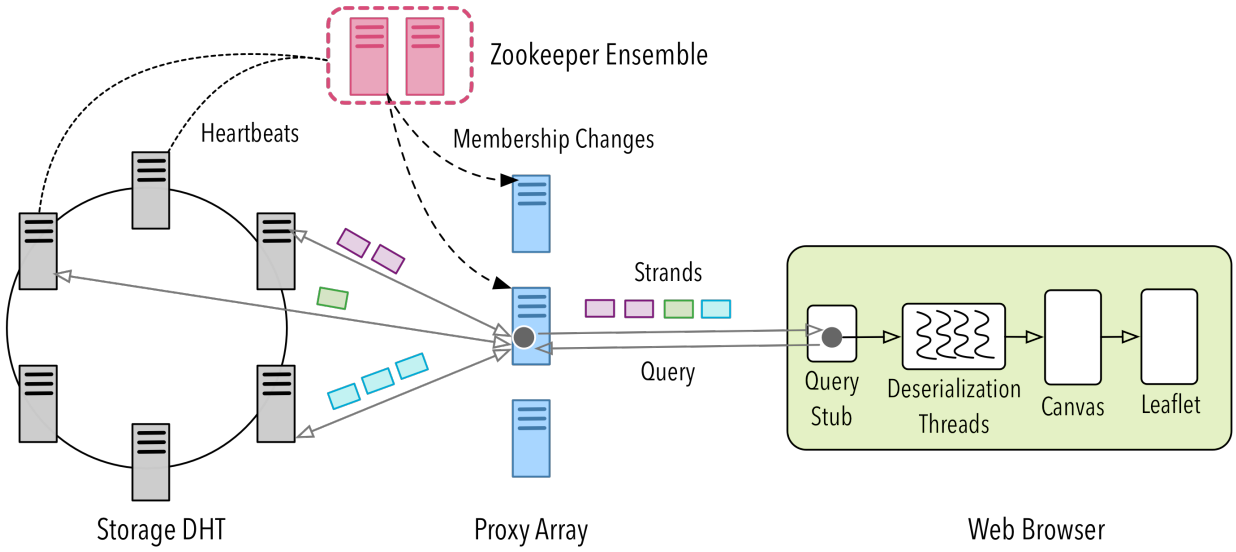

Figure 3.5: The high-level architecture of our query model. Proxy servers are responsible for propagating a query to multiple storage nodes, aggregating the response streams into a single stream, and sending the results back to the query client running on a browser. A small Zookeeper cluster is used to identify newly added and failed nodes.

grained temporal component (e.g., the month of the year) to generate the key for data dispersion. Using a coarser-grained temporal component, instead of the higher resolution timestamps, provides better temporal locality during data storage. Consistent hashing on a key generated by combining both spatial and temporal components and the use of virtual nodes [51] enables better load balancing across the DHT.

At each DHT node, strands are indexed both spatially and temporally for efficient retrieval. Geohashes are indexed using a prefix tree to support efficient wildcard matching. Leaf nodes of the prefix tree are log structured merge (LSM) trees where strands are stored in a sorted order based on their timestamps. Each LSM tree contains a hierarchical temporal index to retrieve matching strands based on temporal predicates of a query. We utilize multiple disks available on a machine for storage to facilitate parallel disk I/O.

During query evaluation, a query is transformed into multiple sub-queries by partitioning the spatial scope. These sub-queries are then evaluated in parallel to leverage the multi-core architecture in the underlying hardware. We also leverage disk caches by pinning frequently accessed data in memory, a technique used in blob-store implementations [52]. Memory limits are enforced on

the storage node processes such that the operating system can use sufficient physical memory for the disk cache. Parallel query execution works with caching and parallel disk I/O to reduce the query latencies significantly.

When performing server-side rendering several additional operations need to be performed. These operations are divided up into three phases:

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 converted into a raster which is sent to the client.

Chapter 4

Performance Evaluation

4.1 Experimental Setup

The evaluation was performed on a client with 32GB of RAM, an Intel Core i5-9300H 2.40GHz CPU, and a Nvidia 1660GTX GPU. Aperture was accessed via a Chrome browser, and GPU support was disabled during some experiments by disabling WebGL canvas acceleration.

The data storage system comprised 75 nodes (Xeon E5-2620, 32 GB Memory, 4 TB storage) and a varying number of proxy servers (Xeon E5-2620, 64 GB Memory) depending on the experiment, each running Fedora 30 and JDK 1.8.0_251. Each machine is connected to the cluster using a 1 Gbps link.

4.2 Responsiveness

Aperture's responsiveness in both server-side and client-side rendering is defined as the time between a query being initiated by a user and the results of the query being visualized. In the server-side configuration, the size of the area being queried is varied and response times are compared. For incremental client-side rendering, the time required to visualize N strands is measured, from which it's possible to extrapolate response times for any area of strand density.

4.2.1 Server-Side Rendering

In Figure 4.1 we report the time required for each phase based on the area being queried. From Figure 4.1, 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.

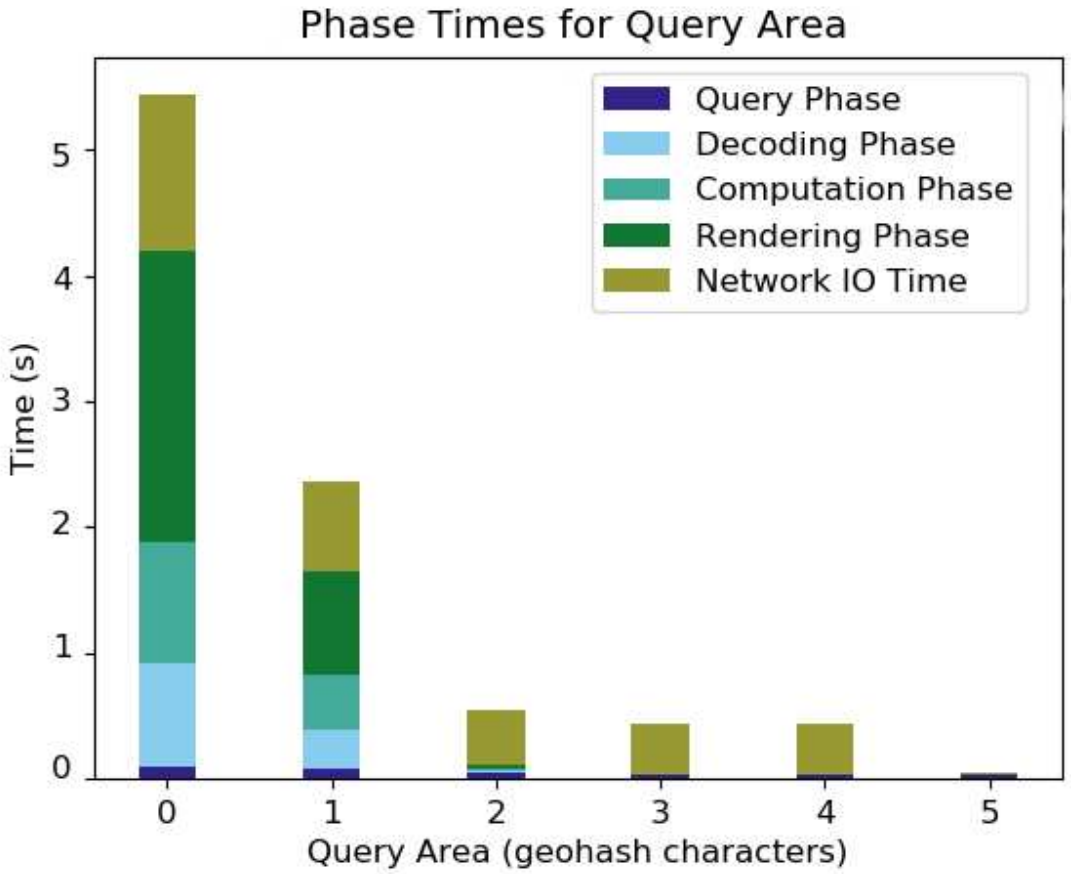

Figure 4.1: A comparison of query times over different query areas. Network and query phase time are relatively constant, while the decoding, computation, and rendering phases dominate query time for large areas.

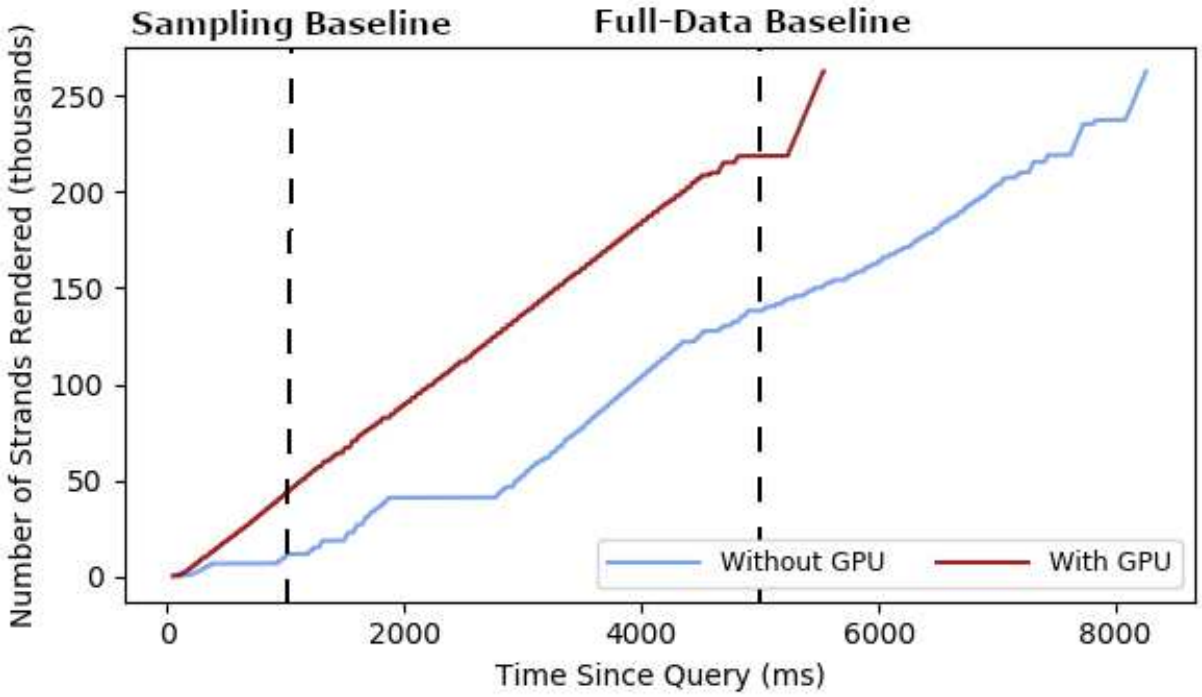

Figure 4.2: A comparison between query times with and without GPU support. Using a GPU, Aperture is able to render data at a faster and more consistent rate.

Each 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.

4.2.2 Client-Side Rendering

Responsiveness with client-side rendering is difficult to quantify due to the incremental nature of the visualization construction. It is also difficult to compare to other systems as there are few specialized geospatial data visualization tools, none incremental. GeoSparkViz [53] is capable of rendering geospatial scatter-plots and heat-maps at a rate of 20 million points per minute, moderately faster than Aperture, however doing so requires the visualization to be generated server-side using all the cores in a Spark cluster, restricting it to one user at a time, and suffers from the overheads of submitting jobs and collecting results, which make sub-second latencies impossible. Interactive geospatial visualization systems, such as Waldo [54], are capable of rendering

continent-scale visualizations in 5-7 seconds. Other visualization systems, such as ISOS [55], achieve real-time interactivity with sampling algorithms. As a heuristic for comparing Aperture's incremental data visualization to the different types of geospatial visualization tools, we define two baselines: the sampling baseline, at one second, and the full-data baseline, at five seconds.

As can be observed in Figure 4.2, the first data points arrive a fraction of a second after a query is issued. Although Figure 4.8 shows that a visualization is not completed immediately, the rapid feedback is likely to improve user perception of the responsiveness of the system [56], as is the visible and constant improvement as the visualization is refined [57]. We can also see that, with co-processor support, Aperture is able to match the full-data baseline using exclusively client-side rendering.

Figure 4.2 also quantifies the difference between running Aperture with and without a GPU. The presence of a co-processor dramatically improves the performance of Aperture, reducing the time to complete a visualization by more than 30%. Additionally, it stabilizes the rate at which the visualization is incrementally rendered, likely because the CPU faces contention from threads responsible for receiving and deserializing data strands.

Table 4.1: Comparison in neural network evaluation times with and without GPU support, averaged over 20 runs.

Processor	Mean Evaluation Time (ms)	Standard Deviation
CPU	2096.75	64.67
GPU	231.60	135.04

4.2.3 Deep-Learning Assisted Rendering

We also evaluate the responsiveness of queries rendered using the pre-trained deep neural network. The results of that evaluation are shown in Table 4.1. The presence of a GPU improves performance by nearly a factor of ten, and, similarly to the incremental rendering operations, per-

forming the evaluation on a GPU reduces to competition between simultaneous rendering and deserialization operations.

4.2.4 Low-Latency Data Retrieval

We have designed our data storage system and the associated query processing system to facilitate low latency and scalable query evaluation. We validate our design decisions outlined in Section 3.4 using a set of micro-benchmarks and system benchmarks.

Given that strands are disk-bound data, we leverage an array of disks attached to a node to provide higher disk I/O (both reads and writes). Figure 4.3 depicts the cumulative disk read throughput achievable with multiple disks attached to a single node. Near-linear increase in the cumulative read throughput implies the minimal overhead at the data access and query processing layer. In our experimental setup, the network bandwidth is the most constrained resource, which indirectly limits the disk read throughput.

Figure 4.4 and Figure 4.5 demonstrate the effect of parallelized query processing and disk caching. At every DHT node, a query is split into a series of subqueries based on the qualifying spatial scopes, which are then processed in parallel. This approach is most effective with larger geospatial scopes — query completion times are improved by 74.7% for geohash prefixes of length 0, whereas the improvement is 38.9% for geohash prefixes with length 4. On the other hand, we observed a consistent improvement, 96.4% - 98.3%, due to disk cache irrespective of the geohash prefix length, as shown in Figure 4.5.

We profiled the query performance of our storage subsystem — the results are depicted in Figure 4.6. Query completion times (latencies) and the cumulative query throughput of the system is recorded for a different number of concurrent queries. Each query retrieved one month's worth of data for an area represented by a geohash prefix of length 4, therefore more uniformly distributing the queries throughout the DHT. Our observations align with the behavior of a typical distributed system approaching the peak performance. Latency stays constant initially before increasing with the cumulative query throughput. Once the query throughput reaches the maximum capacity, the

latency increases exponentially. Because the data storage system is the only aspect of Aperture shared between users, all rendering is pushed to the client, this demonstrates that our experimental setup can handle as many as 100 simultaneous queries before reaching capacity. Because users are unlikely to be making queries every half-second, this translates to thousands or tens of thousands of simultaneous clients.

Proxies act as the interface between the query clients and the backend DHT, therefore they tend to become a bottleneck under heavy load. More proxy servers are provisioned on-demand to counter this issue. This behavior is demonstrated in Figure 4.7 — we saturate the network bandwidth of a single proxy server using a set of query clients operating simultaneously. As more proxy servers join the array, the cumulative query output (measured in terms of data transfer rate) is increased, while latencies are improved significantly. We do not see a linear improvement of performance because we keep the initial load unchanged, which is not sufficient to saturate the network bandwidths of multiple proxy servers.

4.3 Fidelity

Aperture is able to generate visualizations with sub-second latency due to a decaying-size rendering scheme in which early data strands are used to approximate the values of spatially adjacent data strands until those strands arrive at the client. This results in a loss in visualization fidelity, as illustrated in Figure 4.9. Both exponential and linear decay schemes are never able to reach optimal fidelity due to approximations made with early data strands. In Figure 4.8 it can be seen that the loss in fidelity occurs around the edges of the dataset where data is scarce. In these regions certain strands are made to approximate data that does not exist, creating some areas of the visualization that will never be refined.

The response time gain, however, is enormous, with the exponential decay scheme reaching its maximum fidelity in a quarter of a second, easily under both the sampling and full-data baselines. Assuming that the area being visualized is not too close to the spatial edge of the dataset, the small

loss in fidelity is visually insignificant (see Figure 4.8) compared to the improvement in response time.

Fidelity of Learned Visualizations

Figure 4.9 also shows the quality of the pre-trained visualizations that are rendered as more data arrives. All the strands that have arrived so far are evaluated and a learned visualization is generated every half-second, as the model evaluation is too slow to be done every time a new strand arrives. In Figure 4.9, the generated raster shows moderately higher error than any of the decay schemes, although it does outperform manual rastering when just a few strands have arrived due to the fact that, even with very little data, the model "knows" where data points are located and the relative temperatures of varying regions.

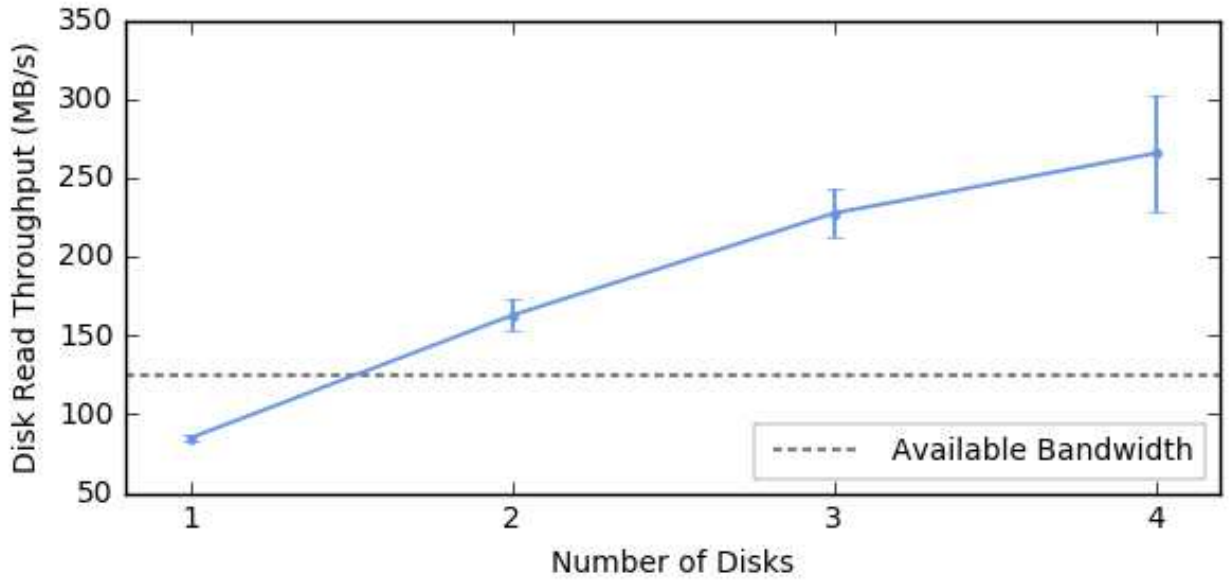

Figure 4.3: Disk read rates at a single node with multiple disks. Near linear growth suggests a low overhead from the data access layer.

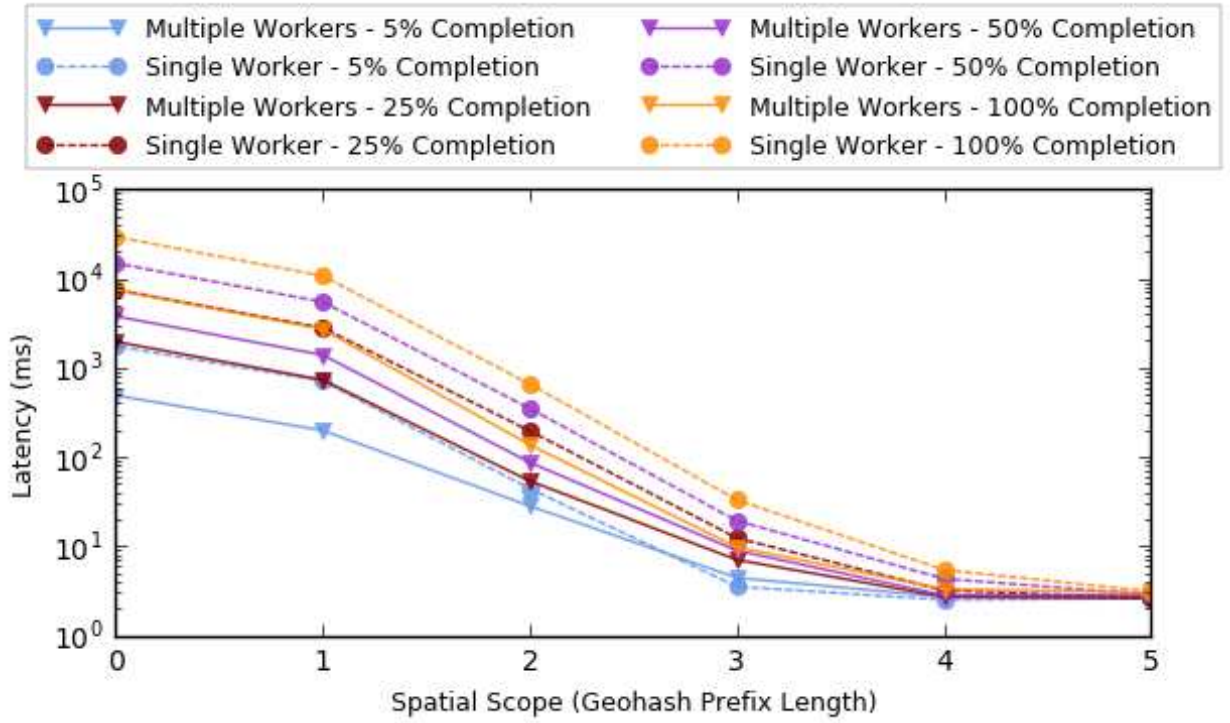

Figure 4.4: Impact of processing a query in parallel using multiple workers. Parallel processing significantly reduces the latency of queries with predicates corresponding to larger scopes.

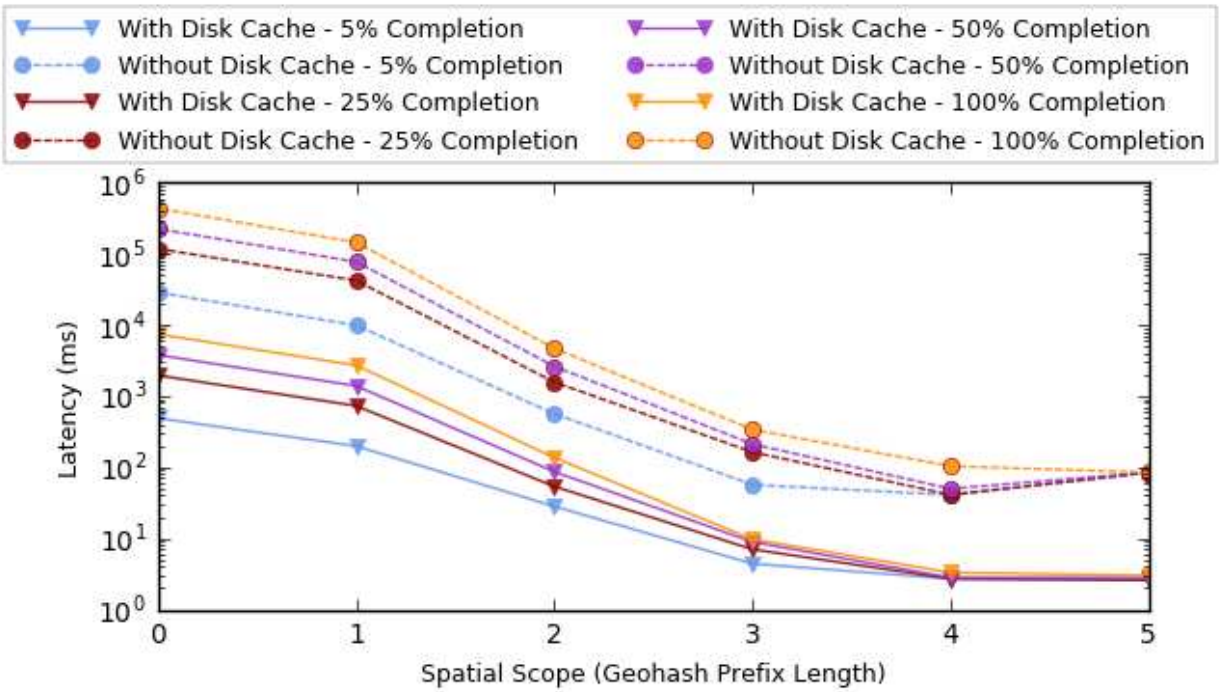

Figure 4.5: Our query processing model extensively relies on disk cache to reduce disk I/O. Using of caching reduces the query latency irrespective of the query scope.

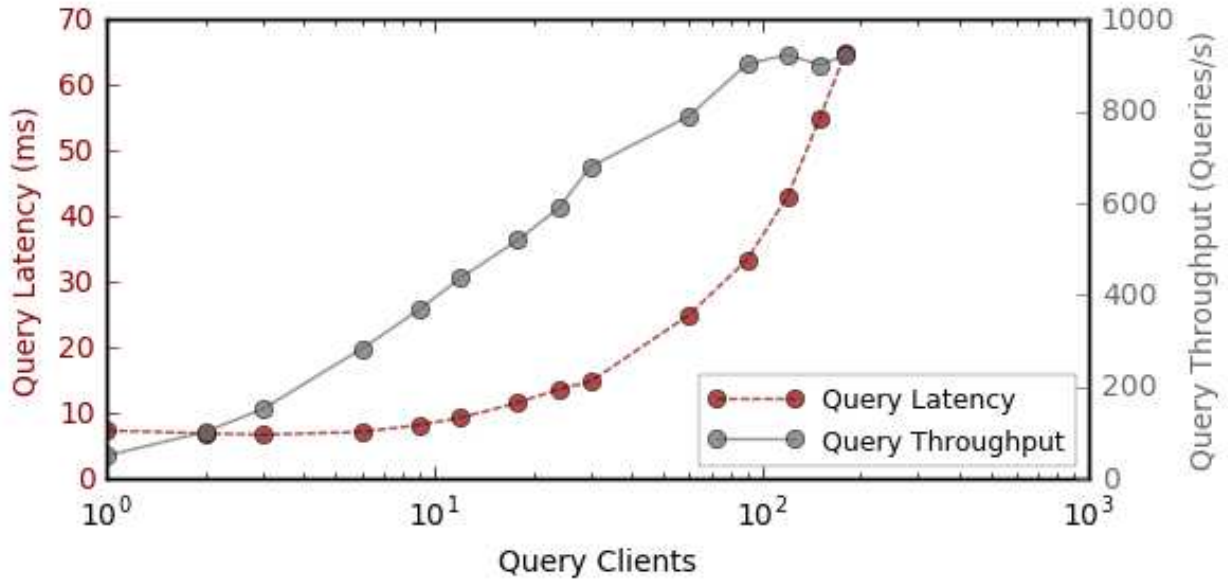

Figure 4.6: Query latency vs. throughput as we funnel multiple queries simultaneously through a single proxy node. Once the proxy nodes reach their maximum capacity, latency grows exponentially.

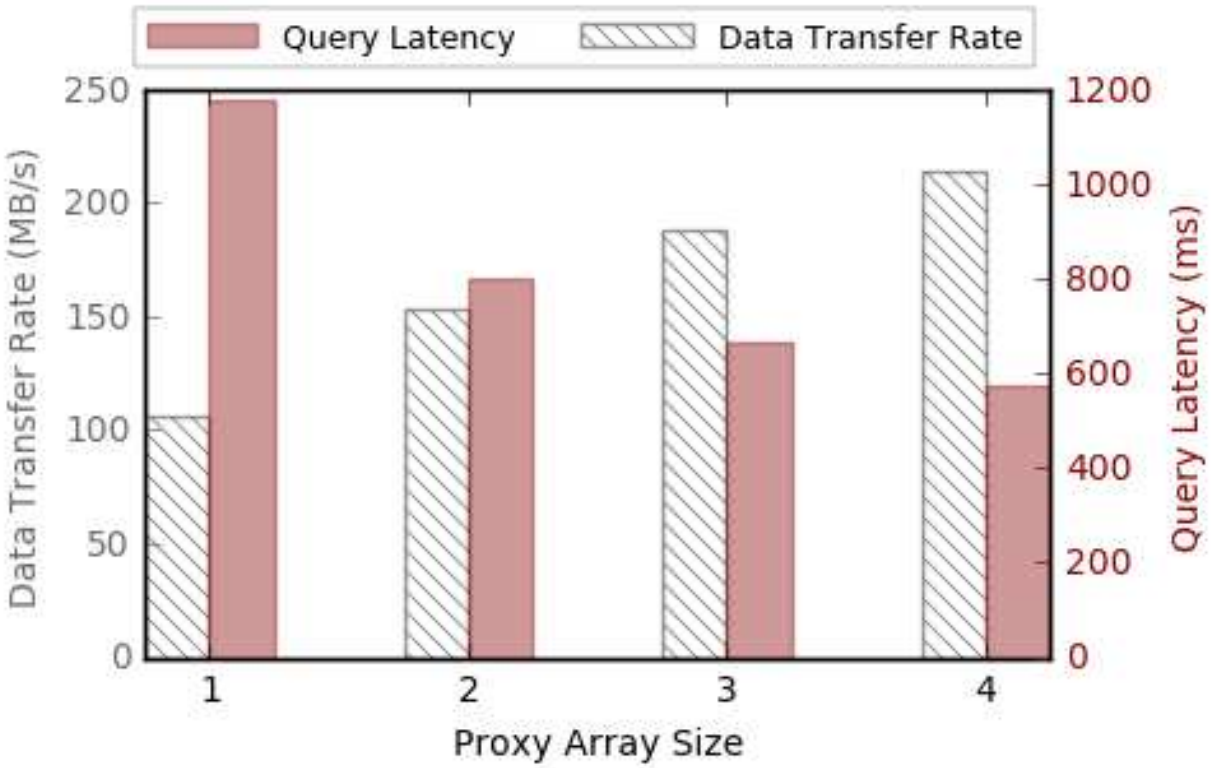

Figure 4.7: Demonstrating the horizontal scalability of the proxy server arrays. Incrementally adding proxy servers relieves the bandwidth bottleneck, improving latency and throughput.

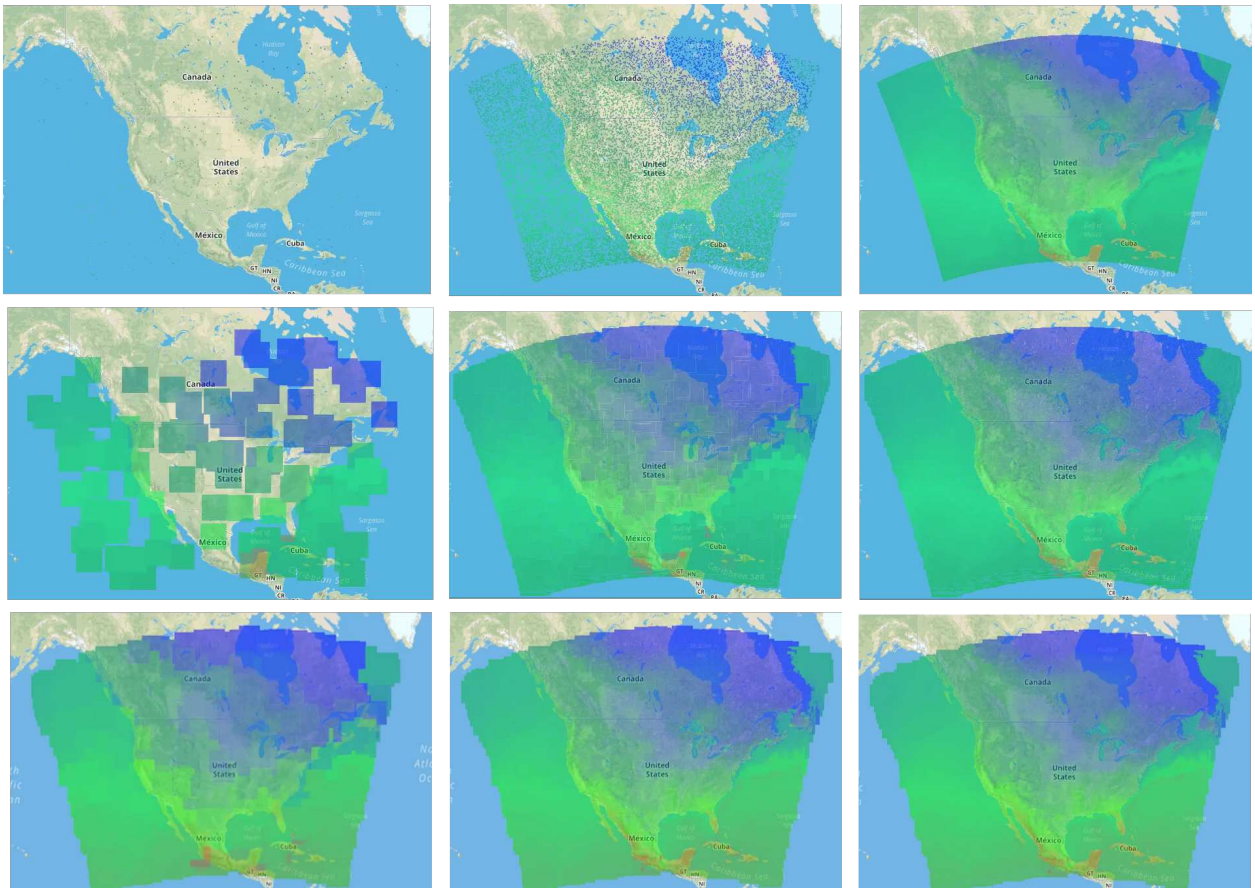

Figure 4.8: Three examples of increasing resolution over time. The top row is without decay, the middle row uses linear decay, and the bottom row uses exponential decay. Visualizations are shown at 1/10th of a second after the query, 3/4^{ths} of a second after the query, and 4 seconds after the query.

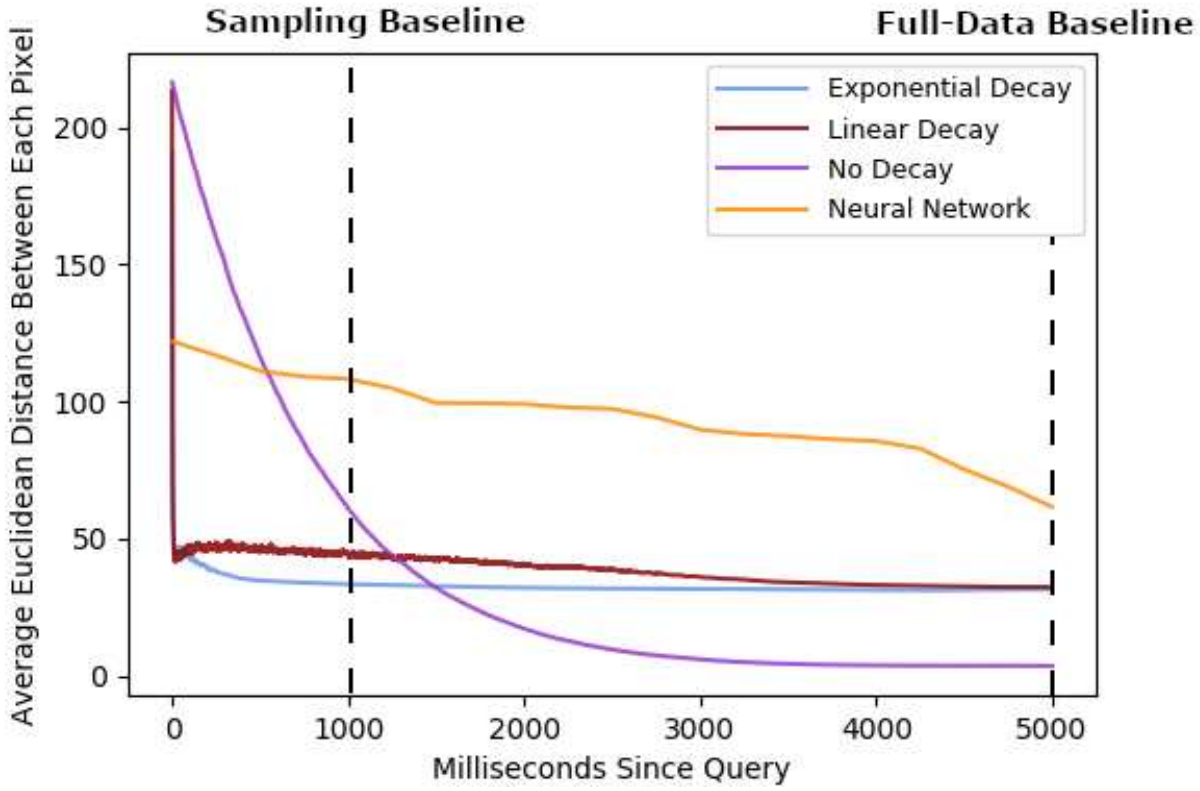

Figure 4.9: A measurement of how image fidelity improves over time with different methods. Comparisons are between the current visualization and the final visualization generated without decay.

Chapter 5

Conclusions

In this study we presented our methodology involving a mix of workload amortizations, forecasts and learning, and exploiting perceptual characteristics to render visual artifacts. In particular, Aperture facilitates interactivity at the client while allowing high-throughput interactions at the server side.

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.

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, is 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.

Our rendering scheme includes a cooperative mechanism involving the client and server side to combine fast coarser-grained rendering with incremental refinements that improves the visualization over time. This is underpinned by our server-side DHT framework that partitions a query into multiple predicates that are not just evaluated concurrently, but also support streaming of results as they become available. Our methodology manages and ensures a high degree of concurrency both in a distributed setting and at a particular node. The degree of concurrency at a node can be dynamically tuned based on the ongoing system load. Besides reducing interference across queries to improve responsiveness, this has the added benefit of high throughput evaluations.

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.

As part of future work, we will explore inter-operation with 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.

Bibliography

[1] Kevin Bruhwiler and Shrideep Pallickara. Aperture: Fast visualizations over spatiotemporal datasets. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, UCC'19, page 31–40, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Kevin Bruhwiler, Thilina Buddhika, Shrideep Pallickara, and Sangmi Lee Pallickara. Iris: Amortized, resource efficient visualizations of voluminous spatiotemporal datasets. In *Proceedings of the IEEE/ACM 6th International Conference on Big Data Computing, Applications and Technologies*, BDCAT'20, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Thilina Buddhika, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2552–2566, 2017.

[4] B. Delaunay. 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*, pages 793–800, 1934.

[5] C. Stolte, D. Tang, and P. Hanrahan. Multiscale visualization using data cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):176–187, April 2003.

[6] Dan Murray. *Tableau Your Data! Fast and Easy Visual Analysis with Tableau Software*. Wiley Publishing, 1st edition, 2013.

[7] J. Koontz, M. Malensek, and S. Pallickara. Geolens: Enabling interactive visual analytics over large-scale, multidimensional geospatial datasets. In *2014 IEEE/ACM International Symposium on Big Data Computing*, pages 35–44, Dec 2014.

[8] Daniel A. Keim, Jörn Schneidewind, and Mike Sips. Circleview: A new approach for visualizing time-related multidimensional data sets. In *Proceedings of the Working Conference on*

Advanced Visual Interfaces, AVI '04, page 179–182, New York, NY, USA, 2004. Association for Computing Machinery.

[9] J. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. C. Law, and M. Papka. Large-scale data visualization using parallel data streaming. *IEEE Computer Graphics and Applications*, 21(4):34–41, 2001.

[10] O. Daae Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *2011 IEEE Pacific Visualization Symposium*, pages 171–178, 2011.

[11] K. Maeda. Performance evaluation of object serialization libraries in xml, json and binary formats. In *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, pages 177–182, 2012.

[12] Tony Parisi. *WebGL: Up and Running*. O'Reilly Media, Inc., 1st edition, 2012.

[13] Steve Fulton and Jeff Fulton. *HTML5 canvas: native interactivity and animation for the web*. " O'Reilly Media, Inc.", 2013.

[14] Danyel Fisher, Igor Popov, Steven Drucker, and m.c. schraefel. Trust me, i'm partially right: Incremental visualization lets analysts explore large datasets faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 1673–1682, New York, NY, USA, 2012. Association for Computing Machinery.

[15] J. Im, F. G. Villegas, and M. J. McGuffin. Visreduce: Fast and responsive incremental information visualization of large datasets. In *2013 IEEE International Conference on Big Data*, pages 25–32, 2013.

[16] Mike Barnett et al. Stat! an interactive analytics environment for big data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, page 1013–1016, New York, NY, USA, 2013. Association for Computing Machinery.

[17] Héléna A. Gaspar, Igor I. Baskin, Gilles Marcou, Dragos Horvath, and Alexandre Varnek. Chemical data visualization and analysis with incremental generative topographic mapping: Big data challenge. *Journal of Chemical Information and Modeling*, 55(1):84–94, 2015. PMID: 25423612.

[18] Michael Glueck, Azam Khan, and Daniel J. Wigdor. Dive in! enabling progressive loading for real-time navigation of data visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 561–570, New York, NY, USA, 2014. Association for Computing Machinery.

[19] Jiawei Jiang, Fangcheng Fu, Tong Yang, and Bin Cui. Sketchml: Accelerating distributed machine learning with data sketches. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1269–1284, New York, NY, USA, 2018. ACM.

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

[21] Q. Huang and P. P. C. Lee. 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*, pages 1420–1428, April 2014.

[22] Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. Sketch-based geometric monitoring of distributed stream queries. *Proc. VLDB Endow.*, 6(10):937–948, August 2013.

[23] Dimitris Papadias, Yufei Tao, P Kanis, and Jun Zhang. Indexing spatio-temporal data warehouses. In *Proceedings 18th International Conference on Data Engineering*, pages 166–175. IEEE, 2002.

[24] Yufei Tao, G. Kollios, J. Considine, F. Li, and Dimitris Papadias. Spatio-temporal aggregation using sketches. In *Proc. of the Intl. Conference on Data Engineering*, pages 214–225, March 2004.

[25] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[26] Sangmi Lee Pallickara, Shrideep Pallickara, and Marlon Pierce. *Scientific Data Management in the Cloud: A Survey of Technologies, Approaches and Challenges*, page 517. 2010.

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

[28] M. Malensek, S. L. Pallickara, and S. Pallickara. Galileo: A framework for distributed storage of high-throughput data streams. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 17–24, 2011.

[29] M. Malensek, S. Pallickara, and S. Pallickara. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing*, 5(1):28–42, 2017.

[30] M. Malensek, S. Pallickara, and S. Pallickara. Analytic queries over geospatial time-series data using distributed hash tables. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1408–1422, 2016.

[31] M. Malensek, S. Pallickara, and S. Pallickara. Polygon-based query evaluation over geospatial data using distributed hash tables. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 219–226, 2013.

[32] Saptashwa Mitra, Paahuni Khandelwal, Shrideep Pallickara, and Sangmi Lee Pallickara. Stash: Fast hierarchical aggregation queries for effective visual spatiotemporal explorations. In *(To appear) IEEE International Conference on Cluster Computing (CLUSTER)*, Albuquerque, NM, USA, 2019.

[33] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Geometry and proximity constrained query evaluations over large geospatial datasets using distributed hash tables. In *IEEE Computing in Science and Engineering (CiSE). Special Issue on Extreme Data*, volume 16, pages 53–60, 2014.

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

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

[36] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, Dec 2013.

[37] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.

[38] T. Buddhika and S. Pallickara. Neptune: Real time stream processing for internet of things and sensing environments. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1143–1152, 2016.

[39] T. Buddhika, R. Stern, K. Lindburg, K. Ericson, and S. Pallickara. Online scheduling and interference alleviation for low-latency, high-throughput processing of data streams. *IEEE Transactions on Parallel and Distributed Systems*, 28(12):3553–3569, 2017.

[40] Gustavo Niemeyer. Geohash, 2008.

[41] BP Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

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

[43] Mapbox. https://www.mapbox.com/, 2019.

[44] Balearic Island Coastal Observing and Forecasting System. Leaflet.TimeDimension, August 2019.

[45] Adam Paszke et al. Pytorch: An imperative style, high-performance deep learning library. pages 8024–8035. Curran Associates, Inc., 2019.

[46] Andreas Buja, Werner Stuetzle, and Yi Shen. Loss functions for binary class probability estimation and classification: Structure and applications. *Working draft, November*, 3, 2005.

[47] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[48] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2019.

[49] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[50] Yulong Wang and Hariharan Sheshadre. Onnxjs. https://github.com/microsoft/onnxjs, April 2020.

[51] Giuseppe DeCandia et al. Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220, 2007.

[52] Shadi A. Noghabi et al. Ambry: Linkedin's scalable geo-distributed object store. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 253–265, New York, NY, USA, 2016. Association for Computing Machinery.

[53] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. Geosparkviz: a scalable geospatial data visualization framework in the apache spark ecosystem. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2018.

[54] D. A. Keim, C. Panse, M. Sips, and S. C. North. Visual data mining in large geospatial point sets. *IEEE Computer Graphics and Applications*, 24(5):36–44, 2004.

[55] Tao Guo, Kaiyu Feng, Gao Cong, and Zhifeng Bao. Efficient selection of geospatial data on maps for interactive and visualized exploration. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 567–582, New York, NY, USA, 2018. Association for Computing Machinery.

[56] John A Hoxmeier and Chris Dicesare. System response time and user satisfaction: An experimental study of browser-based applications. *Proceedings of the Association of Information Systems Americas Conference*, 01 2000.

[57] Nicola Cranley, Philip Perry, and Liam Murphy. User perception of adapting video quality. *International Journal of Human-Computer Studies*, 64(8):637–647, 2006.