Synopsis: A Distributed Sketch over Voluminous Spatiotemporal Observational Streams in Support of Real-Time Query Evaluations

Thilina Buddhika, Matthew Malensek, Sangmi Lee Pallickara and Shrideep Pallickara, *Members, IEEE*

Abstract—Networked observational devices have proliferated in recent years, contributing to voluminous data streams from a variety of sources and problem domains. These streams often have a spatiotemporal component and include multidimensional *features* of interest. Processing such data in an offline fashion using batch systems or data warehouses is costly from both a storage and computational standpoint, and in many situations the insights derived from the data streams are useful only if they are timely.
In this study, we propose SYNOPSIS, an online, distributed *sketch* that is constructed from incoming observational stream data. The sketch summarizes feature values and inter-feature relationships in memory to facilitate real-time query evaluations. As the data streams evolve and user query patterns change, SYNOPSIS performs targeted dynamic scaling to ensure high accuracy and low query latencies alongside effective resource utilization. We evaluate our system in the context of a real-world spatiotemporal dataset and demonstrate its efficacy in both scalability and query evaluations.

Index Terms—real-time stream processing; online scheduling; data intensive computing

1 INTRODUCTION

The proliferation of remote sensing equipment (such as radars and satellites), networked sensors, commercial mapping, location-based services, and sales tracking techniques have all resulted in the exponential growth of spatiotemporal data. Spatiotemporal data comprise observations where both the location and time of measurement are available in addition to *features* of interest (such as humidity, air quality, disease prevalence, sales, etc). This information can be leveraged in several domains to inform scientific modeling, simulations, resource allocation and decision making. Relevant domains include atmospheric science, epidemiology, environmental science, geosciences, smart-city settings, and commercial applications. In these settings, queries over the data must be *expressive* to ensure efficient retrievals. Furthermore, query evaluations must be executed in real time with low latency, regardless of data volumes.

Spatiotemporal datasets are naturally multidimensional with multiple features of interest being reported/recorded continuously for a particular timestamp and geolocation. The values associated with these features are continually changing; in other words, the dataset *feature space* is always evolving. Queries specified over these datasets may have a wide range of characteristics encompassing the frequency at which they are evaluated and their spatiotemporal scope. The crux of this paper is to support query evaluations over continually-arriving observational data. We achieve this via construction of an in-memory distributed *sketch* that maintains a compact representation of the data.

1.1 Challenges

Support for real-time evaluation of queries and analysis over a feature space that is continually evolving introduces unique challenges. These include:

- *Data volumes:* It is infeasible to store all the observational data. This is especially true if the arrival rates outpace the rate at which data can be written to disk.
- *Data arrival rates:* The data may arrive continually and at high rates. Arrival rates may change over time.
- *I/O Costs:* Memory accesses are 5-6 orders of magnitude faster than disk accesses. Given the data volumes, disk accesses during query evaluations are infeasible.
- *Accuracy:* Queries evaluations must be accurate, with appropriate error bounds or false positive probabilities included in the results.
- *Spatiotemporal characteristics:* Queries may target spatiotemporal properties. For example, a user may be interested in feature characteristics for a geospatial location at a particular daily interval over a chronological range (for example, 2:00–4:00 pm over 2–3 months).

1.2 Research Questions

The challenges associated with accomplishing this functionality led us to formulate the following research questions:

RQ-1: How can we generate compact, memory-resident representations of the observational space while accounting for spatiotemporal attributes? The resulting *sketch* must be amenable to fast, continuous updates to ensure its representativeness and fidelity to the original data.

RQ-2: How can we scale effectively in situations where system load is high or the observations arrive at a rate faster than the rate at which the sketch can be updated? The density and arrival rates for observations may vary based on geospatial characteristics. For example, in a smart-city setting, New York would have a far higher rate of observations than Denver.

RQ-3: How can we enable expressive, low-latency queries over the distributed sketch while also maintaining accuracy? Given that the sketch is a compact representation of the data, queries facilitate high-level analysis without requiring users to understand the underlying system implementation.

- *T. Buddhika, M. Malensek, S.L. Pallickara and S. Pallickara are with the Department of Computer Science, Colorado State University. E-mail: {thilinab, malensek, sangmi, shrideep}@cs.colostate.edu*

1.3 Approach Summary

Similar to other sketches, the design of SYNOPSIS is guided by the functionality that we wish to support. Synopsis is designed to be a compact, effective surrogate for voluminous data. Synopsis extracts metadata from observations and organizes this information to support relational queries targeting different portions of the feature space; we support selection, joins, aggregations, and sorting. The SYNOPSIS sketch can interoperate and provide input data to general purpose computations expressed using popular analytic engines such as Spark [1], [2], TensorFlow [3], [4], Hadoop [5], [6], [7], and VW [8].

Our sketch is also naturally amenable to distribution, with each machine in the cluster holding information about a particular subset of the observational space. This ensures each cluster-node in the system can evaluate multiple concurrent queries independently. The sketch is capable of scaling in or out depending on streaming ingress rates and memory footprints, with scale-out operations that support targeted alleviation of hotspots. Our framework manages the complexity of identifying these hotspots, splitting portions of the sketch, and migrating the relevant subsets to nodes with higher capacity. Distributing the sketch across multiple cluster-nodes allows us to maintain a finer-grained representation of the feature space while also improving the accuracy of query evaluations; for example, an arctic region and a tropical region would be maintained on separate nodes that specialize for particular climates.

To our knowledge, SYNOPSIS is the first sketch designed specifically for spatiotemporal observational data. We have validated the suitability of our approach through a comprehensive set of benchmarks with real observational data.

1.4 Paper Organization

The remainder of this paper is organized as follows. Section 2 provides an overview of the system, followed by our methodology in Section 3. We present performance evaluation results of our system in Section 4. Section 5 demonstrates applications of SYNOPSIS, Section 6 discusses related approaches, and Section 7 concludes the paper and outlines our future research direction.

2 System Overview and Preliminaries

SYNOPSIS is a distributed sketch constructed over voluminous spatiotemporal data streams. The number of sketchlets (executing on different machines) that comprise the distributed sketch varies dynamically as the system scales in or out to cope with data arrival rates and memory pressure. SYNOPSIS executes a set of stream ingesters that reads data off the spatiotemporal data streams and inject into the sketch. A stream partitioning scheme, based on the Geohash algorithm (described below), is used to route packets to the appropriate sketchlet. Sketchlets processes stream packets emitted by the stream ingesters and construct compact, in-memory representations of the observational data by extracting metadata from stream packets. During dynamic scaling operations, the geographical extents managed by a sketch varies.

Geohash Algorithm

We use the Geohash algorithm [9] to balance load and partition incoming data streams. Geohash divides the earth into a hierarchy of bounding boxes identified by Base 32 strings; the longer the geohash string, the more precise the bounding box. Figure 1 illustrates this hierarchy. Most of the eastern United States is contained within the bounding box described by geohash *D*, while *DJ* encompasses substantial parts of Florida, Georgia, and Alabama. The bounding box *DJKJ* (highlighted in red) contains Tallahassee, Florida. This hierarchical representation enables SYNOPSIS to cope with both low- and high-density regions: several cluster-nodes may be tasked with managing streams originating in and around large cities, while rural areas fall under the purview of a single node.

To achieve fine-grained control over our geohash partitions, we operate at the bit level rather than Base 32 character level when routing streams. Each bit added to a geohash string reduces its scope by half, with each character represented by five bits ($2^5 = 32$). In other words, a four-character geohash string represents 20 spatial subdivisions applied recursively to each resulting region. This property allows us to manage and allocate resources across a wide variety of observational densities.

SYNOPSIS relies on a set of auxiliary services that are needed to construct, update, and maintain the sketch and also to adapt to changing system conditions:

Control plane: The control plane is responsible for orchestrating control messages exchanged between SYNOPSIS nodes as part of various distributed protocols such as dynamic scaling. It is decoupled from the generic data plane to ensure higher priority and low latency processing without being affected by buffering delays and backpressure experienced during stream processing.

Gossip subsystem: While a majority of the SYNOPSIS functionality relies on the local state constructed at a particular node, certain functionalities require approximate global knowledge. For instance, each sketchlet maintains a geohash prefix tree to assist in distributed query evaluations by forwarding queries to sketchlets that are responsible for particular geographical extents. In order to establish and maintain this global view of the entire system, sketchlets gossip about their state periodically (based on time intervals and the number of pending updates) as well as when a change in state occurs. SYNOPSIS supports *eventual consistency* with respect to these updates given their inherent propagation and convergence delays.

Querying subsystem: The querying subsystem is responsible for the distributed evaluation of queries. This involves for-

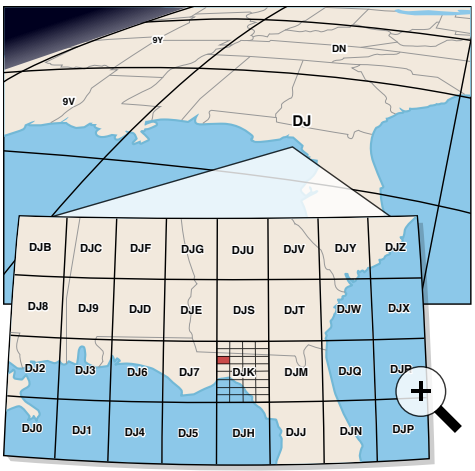


Fig. 1: A demonstration of the Geohash algorithm. Each additional character in a geohash describes a finer-grained region; geohash *DJ* contains substantial parts of Florida, Georgia, and Alabama, USA, while *DJKJ* (highlighted in red) encompasses Tallahassee.

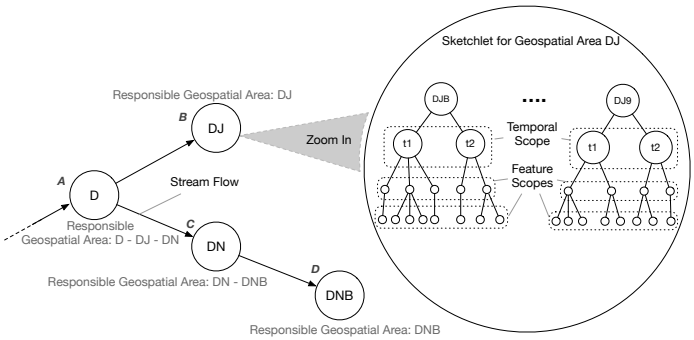


Fig. 2: Demonstration of the distributed sketch for geospatial region with the geohash prefix D. The sketchlets for geohash prefixes DJ and DN have scaled out due to high volume of observations. Each sketchlet maintains a SIFT data structure, which is a forest of trees where each tree is responsible for a more specific geospatial region.

warding queries to relevant sketchlets; in some cases, multiple sketchlets may be involved based on the geographical scope of the query.

Monitoring subsystem: Sketchlets comprising SYNOPSIS are probed periodically to gather metrics that impact performance of the system. These include memory utilization and backlog information based on packet arrival rates and updates to the in-memory structures. This information is used for dynamic scaling recommendations as explained in Section 3.4 and Section 3.5.

3 METHODOLOGY

In this section, we discuss about the construction of the distributed sketch, sketchlet data structure, dynamic scaling, and query support in SYNOPSIS. In most of those subsections, we present microbenchmarks which were run using a single machine (HP DL160 server with Xeon E5620 processor and 12 GB RAM) demonstrating the efficacy of individual aspects of the system. Our input data was sourced from NOAA North American Mesoscale (NAM) Forecast System [10].

3.1 Sketch

The macroscopic view of the distributed sketch is one that comprises multiple sketchlets; each sketchlet executes on a different machine and is responsible for organizing data for a particular geospatial scope. The sketch is organized as a distributed prefix tree. All the descendant nodes – the sketchlets – share a common prefix associated with the parent. Within a sketchlet all observations share the prefix associated with that sketchlet.

The sketch must be performant and flexible while being amenable to scaling operations. The sketch initiates scale-out operations to relieve memory pressure and preserve performance in the face of voluminous data. Scale-in operations are initiated by the sketch to conserve memory. Any sketchlet may serve as the conduit for incoming queries or analytic operations over the sketched spatiotemporal data: the sketch must be organized such that the sketchlets are not involved in redundant query evaluations or analytic operations.

The geohash algorithm is well suited to our problem and plays a central role in the organization of the distributed sketch. Since the geohash algorithm represents a bounding box, it facilitates collation of observations from a particular geographical scope. This in turn allows us to redirect queries effectively and ensure data locality during query evaluations. Increases in the length of the geohash string correspond to geographically smaller bounding boxes being identified with increasing precision. This is also well-aligned with dynamic scaling maneuvers performed by the sketch to manage memory requirements and the volume, rate and density of observational data. Scaling operations within the sketch are targeted. Scale-out operations target geospatial locations with increased density of observations to relieve memory pressure and alleviate performance bottlenecks. Scale-in operations target geolocations where there is a sparsity in available data to conserve memory.

Each sketchlet is responsible for real-time organization, summarization, and compaction of observational data from the geographical scope represented by the sketchlet's geohash. The sketchlet performs two operations. First, it extracts metadata from incoming observations. Metadata extracted from individual observations include: geolocations, chronological information, and features encapsulated within the observation. Second, the sketchlet is responsible for summarization and compaction of measurements and accompanying spatiotemporal information extracted in the previous step.

The sketchlet organizes its summarization of the observational data, as a forest of trees in a data structure called SIFT (Summarization Involving a Forest of Trees). The edges and vertices within each SIFT tree maintain inter-feature relationships, while leaves contain online, in-memory summary statistics and correlation information to support statistical queries and generation of synthetic datasets. The number of edges at each level within the subtrees corresponds to density-based dynamic binning of a particular feature to support error reduction during query evaluations. The underlying principle within this data structure is **grouping** to exploit similarities in values reported within observations. This organization principle extends to all dimensions associated with the observations: spatial, temporal, and encapsulated features. The grouping principle allows us to preserve fidelity of the observational space while conserving memory footprints.

A simplified version of the distributed sketch for geospatial region D is depicted in Figure 2. Each tree within the SIFT is rooted at a higher precision geohash than that associated with the sketchlet. For example, at a sketchlet with a geohash prefix, *DJ*, the trees within the SIFT at that sketchlet are rooted at higher precision geohashes such as *DJB*, *DJC*, *DJF*, etc. An advantage of this approach is that the sketchlet partitions data from different regions within the larger geospatial scope into smaller regions. This partitioning allows the data structure to further exploit similarity in the observation values reported for that spatial scope.

Within each SIFT, the second level is used to group observations based on their temporal properties. This approach allows us to exploit similarity in readings reported for a particular time range. Note that as the trees are traversed, this organization strategy means that all descendants of a temporal node correspond to measurements reported for a particular region and for a particular temporal scope. The SIFT data structure also supports finer-grained temporal resolutions for the recent past – e.g., minutes, hours, day, weeks, etc. – along with targeted compaction operations that fold finer-grained temporal scopes into a coarser grained scopes as time advances. Specifically, our organizational structure allows us to support varying levels of expressiveness for different temporal scopes, with recent observations being represented more expressively. For example, on 3/2/2017 we may maintain subtrees at the minute level for 3:01 pm, 3:02 pm, etc., at 3/2/2017 @ 5:00 pm these subtrees

will be folded into observations for the hourly temporal scope for 3:00-4:00 pm, and at 4:00 pm the next day (3/3/2017) these would then be folded into the coarser gained temporal bin for the previous day.

The grouping concept also extends to individual features. Each feature occupies a level within an individual tree in SIFT. At each level, the range of values that a feature can take is broken up into a set of bins (corresponding to the range of values) that they take. These ranges are determined using kernel density estimation (KDE) to ensure that the binning of features is representative of the observed density in the distribution of values for that feature at the particular spatiotemporal scope. Each node (or bin) maintains the min, max, standard deviation, mean, and the total number of observations it is responsible for. This is useful during the creation of synthetic datasets that are representative of the observational space for a particular spatiotemporal scope.

Our methodology of grouping and organizing the summarization information as a forest of trees accomplishes two key objectives. First, it captures the distribution of feature values across a spatiotemporal scope. Second, it supports targeted reductions in the observational data volumes while being representative of the observed feature values. This is in contrast to a random sampling scheme, which may be unable to recreate distributions with high fidelity for arbitrary spatiotemporal scopes or may drop significant values.

The organization of the sketchlet is such that it is amenable to scale-out and scale-in operations of the distributed sketch. A key feature provided by the SIFT data structure is support for scaling operations. For example, if a subregion represented by a tree within the forest maintained at each sketchlet has a higher density (and variability) of the reported observational values, that tree would have a correspondingly higher memory footprint within the data structure. This allows us to target scaling maneuvers to particular subregions managed at a sketchlet to alleviate memory pressure. During scale-in operations, descendants can be folded into the parent; the descendant's SIFT is simply added as a tree to the SIFT maintained at the parent.

3.2 Data Model

Individual observations are geotagged and have chronological timestamps indicating where and when the observations were made. Location information is encoded as a $\langle latitude, longitude \rangle$ tuple. Observations are multidimensional with multiple features (e.g, temperature, humidity, wind speed, etc.) being reported in each observation. These features may be encoded as $\langle feature\_name, value \rangle$ tuples or may have predefined positions with the observational data's serialized representation.

SIFT: The SIFT data structure is organized as a forest of trees. Data encapsulated within the observations are used to populate trees within SIFT. Each tree within SIFT has the geocoding information as its root node, temporal information at the second level, and individual features encapsulated within the observations at successive levels. The system may shuffle positioning of the features to conserve memory or speed up evaluations for dominant queries.

Additional nodes may be introduced to the trees within a SIFT to collate observations that fall within a certain range. In the case of chronological information these nodes correspond to different temporal scopes such as hours, days, weeks, months, and years. In the case of individual features, these nodes represent a binning of the feature values that correspond to the density in the distribution of observed feature values. Each feature node stores auxiliary information such as min, mean, max, standard deviation, and number of observations to capture statistical aspects about the distribution of values within the bin. Summary statistics are updated in an online fashion and keep pace with data arrival rates.

Systems View of the Sketch: The Synopsis sketch, comprising sketchlets dispersed over multiple machines, is a compact and memory-resident surrogate for the entire observational space. The sketch may be used for any purpose that regular, on-disk data is used for including but not limited to: query evaluations, assessing statistical properties of the data, and launching custom computations using well-known analytical engines. The system imposes no restrictions on the programming language or the type of processing logic that may be encoded within the analytical computations.

The Synopsis sketch is adaptive and evolves over time. The number of sketchlets comprising the sketch varies as the sketch performs scaling maneuvers to cope with data volumes and memory management. The number of trees within the SIFT also varies over time as temporal scopes are aggregated, features binned, and sketchlets are spawned and fused during scaling operations.

3.3 Sketchlet

The distributed sketch maintained by SYNOPSIS is composed of many *sketchlets* dispersed over a cluster of machines. Sketchlets maintain compact, multidimensional, tree-based representations of incoming data streams in a data structure called SIFT (Summarization Involving a Forest of Trees). Each in-memory SIFT can be queried to retrieve statistical properties about the underlying data or discover how features interact. Due to the voluminous nature of these data streams, storing each individual record in main memory is not practical. Therefore, the queries supported by our framework are facilitated by compact, online metadata collection and quantization methods. These techniques ensure high accuracy while also conforming to the memory requirements of the system. To further improve accuracy, we bias our algorithms toward the most recent data points while reducing the resolution of the oldest.

3.3.1 SIFT Structure

SIFT instances are maintained as hierarchical trees with feature values stored in the vertices. Each level of the hierarchy, called a *plane*, represents a particular *feature type*, and traversing through vertices in this feature hierarchy reduces the search space of a query. Upon insertion of a multidimensional observation, each feature is arranged to form a *path* through the tree and added based on the current feature hierarchy. Paths taken through the tree during a lookup are influenced by the specificity of the query, with additional feature expressions constraining the *query scope*; an empty query would result in a scope that spans the entire tree. Figure 3 demonstrates the structure of a tree within a SIFT and highlights a query and its scope. Note that any subset of the tree can be retrieved and manipulated in the same manner.

Metadata records for paths through the feature hierarchy are stored at leaf nodes. Each record contains statistics that are updated in an online fashion using Welford's method [11]. Welford's method maintains the number of observations, n, the running mean, $\bar{x}$, and the sum of squares of differences from the current mean, S_n, demonstrated by the following recurrence relation:

$$\bar{x}_0 = 0, S_0 = 0$$
$$\bar{x}_n = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$
$$S_n = S_{n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)$$

Besides the observation count and running mean, this enables calculation of the variance and standard deviation of the observed values:

$$\sigma^2 = \frac{S_n}{n} \qquad \sigma = \sqrt{\frac{S_n}{n}}$$

Our implementation of Welford's method also includes cross-feature relationships, such as the correlation between temperature values and humidity or the reflectivity of the earth and cloud cover. This is achieved by maintaining the sum of cross products between features. Leaf nodes may be *merged* to combine their respective summary statistics into a single aggregate summary. This allows queries to be evaluated across multiple sketchlets and then fused into a single, coherent result.

The number of unique feature types stored in the SIFT directly influences the size of the hierarchy, impacting memory consumption. However, the number of vertices and edges that must be maintained by each tree can be managed by manipulating the hierarchical configuration. For instance, the memory impact of features that exhibit high variance over a large range can be mitigated by placing them toward the top of the hierarchy, while boolean features or those with low variance should be situated toward the bottom of the tree. Consequently, we *compact* the logical representation of the SIFT by aggregating vertices from the entire forest and reorienting the feature planes to conserve memory. One notable result of this process is that leaf vertices may be responsible for storing spatial locations of the data points rather than the root of the tree because the resulting configuration decreases memory consumption. Feature planes are reconfigured dynamically based on their corresponding vertex *fan-out* during the initial population of the trees. In this phase full-resolution feature values are stored at each vertex, but once a steady state is reached the *quantization* process begins to determine bin sizes and the feature plane hierarchy.

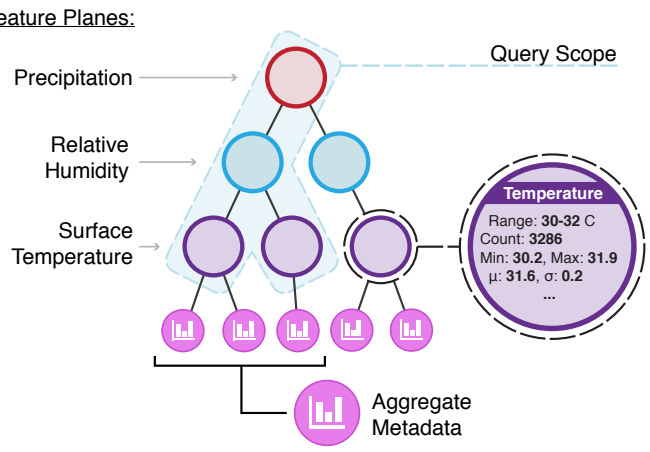


Fig. 3: A simplified SIFT tree with a three-plane hierarchy and sample query scope, leading to several metadata leaves. In production settings, SIFT trees contain hundreds of thousands of vertices and edges.

3.3.2 Density-Driven Quantization

Maintaining data points, statistics, and cross-feature relationships in memory at full resolution is infeasible when faced with voluminous datasets, even when load is balanced over several computing resources. To reduce the memory consumption of SIFT instances we perform *quantization* — targeted reduction of resolution — which allows vertices in the tree to be merged, thus enabling single vertices to represent a collection of values. We determine which vertices should be merged by splitting each range of feature values into a configurable number of *bins*. After quantization, each vertex represents a range of observations.

To determine the size and quantity of these bins, trees within the SIFT maintain additional metadata provided by the multivariate online kernel density estimation (oKDE) algorithm developed by Kristan et al. [12]. While it is possible to recompute kernel density estimates periodically for each feature type using in-memory samples [13], the online approach afforded by oKDE requires less overall CPU usage and memory, which is crucial in streaming environments. oKDE assimilates data incrementally at runtime to create a dynamic probability density function (PDF) for each feature type. The smoothing parameter used to create the PDF, called the *bandwidth*, is selected autonomously using Silverman's rule [14]. Silverman's rule assumes that data tends to follow a normal distribution, which is generally true for naturally-occurring observations. However, we also allow the smoothing parameter be selectively reconfigured for different problem types. During the quantization process, these PDFs are used to ensure that each bin is assigned an approximately equal proportion of the feature density, while the overall number of bins is influenced by memory availability. As a result, the majority of values for a given feature type will be stored in small, highly-accurate bins.

Figure 4 illustrates the quantization process for the *surface temperature* feature in our atmospheric test dataset [10]: the highest densities of values are stored in the smallest bins (indicated by vertical lines under the curve), improving overall accuracy. To evaluate accuracy, we compare the mean values of each bin with the actual, full-resolution data points. Consequently, the *standard error* ($\sigma_{\bar{x}}$) can be calculated from our running summary statistics to judge the accuracy level of the bins based on how well they are represented by the mean:

$$\sigma_{\bar{x}} = \sqrt{\frac{S_n}{n^2}}$$

This information is provided alongside any query results returned by the system. During initialization, we calculate the normalized error for each data point empirically (shown in the lower portion of Figure 4). For values that are observed less frequently, the error rate is higher; temperatures from 240 – 260 Kelvin (-33.15 to -13.15 °C) reach a normalized root-mean-square error (NRMSE) of about 7%. However, approximately 80% of the values in the tree will be assigned to vertices with an error of about 0.5%. In practice, this means that commonly-observed values returned by SYNOPSIS will be within 0.25 Kelvin of their actual value.

Table 1 compares full-resolution and quantized trees that were generated from a month of data with 20 unique features from our test dataset, which includes atmospheric information such as temperature, humidity, precipitation, and cloud cover. In this configuration, our autonomous quantization algorithm reduced memory consumption by about 62.4%, which allows much more historical data to be maintained in each SIFT.

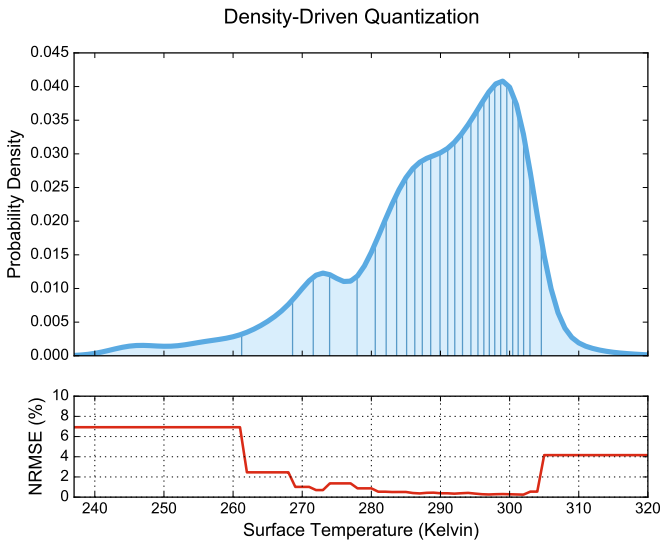


Fig. 4: A demonstration of the quantization process, with 29 vertex bins generated across the distribution of surface temperature values in our dataset. Each bin is indicated by a vertical line under the curve.

Decreasing the memory footprint of the SIFT also allows larger geographical areas to be maintained by a single sketchlet.

TABLE 1: Tree statistics before and after our dynamic quantization algorithm over one month of ingested data.

Metric	Original	Quantized	Change
Vertices	3,104,874	1,238,424	-60.1%
Edges	3,367,665	1,441,639	-57.2%
Leaves	262,792	203,216	-22.7%
Memory	1,710.6 MB	643.1 MB	-62.4%

3.3.3 *Temporal Dimensionality Reduction*

While our quantization approach enables SYNOPSIS to retain large volumes of data in main memory, we also offer a temporal *accuracy gradient* to ensure the most relevant data points are prioritized for high accuracy. This is achieved by iteratively removing tree paths from the SIFT hierarchy in the oldest subtrees, eventually phasing out old records. A user-defined "length of study" (for instance, 12 months) informs the system when dimensionality reduction can begin. As data ages, this process results in the creation of temporal accuracy bands.

Selective temporal dimensionality reduction proceeds in a bottom-up fashion, starting from the leaf nodes. Given a set of relevant vertices, neighboring bins are merged uniformly across the feature space. As the bins are merged, their respective metadata is also merged, reducing memory consumption. Given two metadata instances, merging results in half the memory footprint. However, it is worth noting that this process is irreversible; once metadata has been merged, it cannot be split at a later point in time. As time passes, entire portions of the feature hyperplane are compacted until a single metadata record is left for a particular temporal range. This allows users to still query the summary statistics and models for historical data, but at a lower level of accuracy.

3.4 Coping with High Data Rates: Scaling out

There are two primary approaches to scaling a sketchlet that is experiencing high traffic: *replication* and *load migration*. In replication-based scaling, new sketchlets are spawned during high data arrival rates that are responsible for identical spatial scopes as their originating sketchlet. Assimilation of the newly-created sketchlet involves partitioning inbound streams directed to the original sketchlet. The upstream node (e.g.: stream ingester) is responsible for this partitioning, which may be performed in a skewed fashion with the new sketchlet receiving a larger portion of the inbound stream. Alternatively, inbound streams to the original sketchlet may also be partitioned in a round-robin fashion between the original and the newly-created sketchlet. Using a replication-based scaling with a round-robin style stream partitioning is inefficient with respect to memory consumption because of the possibility of multiple SIFT trees with significantly overlapping sets of vertices and edges.

In targeted load migration, particular geospatial scopes, i.e. SIFT trees are evicted from the original sketchlet to the newly created sketchlet during heavy traffic. Deciding which SIFT trees to migrate is based on data arrival rates and the rates at which particular SIFT trees are being updated.

In SYNOPSIS, we use targeted load migration for scaling out. Our implementation closely follows the MAPE loop [15] which comprises four phases: monitor (M), analyze (A), planning (P) and execution (E). A **monitoring** task within every sketchlet periodically gathers two performance metrics:

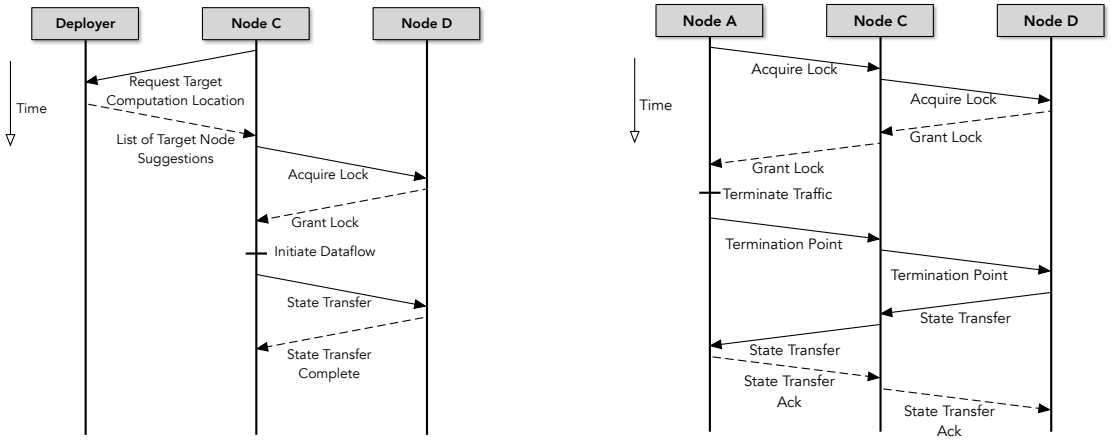
1) **Length of the backlog:** This represents the number of unprocessed messages in the queue. If the sketchlet cannot keep up with the incoming data rate, then the backlog will grow over time.
2) **Memory pressure:** Each sketchlet is allocated a fixed amount of memory. Exceeding these memory limits creates memory pressure, which may cause extended garbage collection cycles, increased paging activity, and will eventually lead to reduced performance. The monitoring task continuously records the memory utilization and triggers scaling activities accordingly.

The objective of scaling out is to maintain the *stability* at each sketchlet. We define stability as the ability to keep up with incoming data rates while incurring a manageable memory pressure. During the **analyze** phase, we use threshold-based rules [16] to issue scale-out recommendations to sketchlets, which are issued if either of the following rules are consistently satisfied for a certain number of monitoring cycles:

- Backlog growth, which indicates that a portion of the load needs to be migrated to a different cluster-node.
- High overall memory utilization above a threshold, which is usually set below the memory limits to allow a capacity buffer for the process to avoid oscillation.

Upon receiving a scaling out recommendation from the monitoring task, the sketchlet executes the **planning** and **execution** phases.

During the planning phase, the sketchlet chooses portion(s) of the region within its current purview, i.e. a set of SIFT trees, to be handed over to another sketchlet. For this task, it relies on performance metrics it maintains for each subregion and a numeric value provided by the scale-out recommendation that measures how much load should be migrated. These metrics includes the data rate and the memory consumption for each subregion. If the backlog growth based threshold rule is triggered the scale out operation, the subregion metrics are sorted based on their update rates in the descending order. Otherwise



(a) Scale-out protocol

(b) Scale-in protocol

Fig. 5: Our dynamic scaling protocol in chronological order.

they are sorted based on their memory consumption. Then a simple bin-packing algorithm is used to choose a minimum set of subregions for migration such that the excess load is removed from the current sketchlet.

Only a single scaling operation takes place at a given time per sketchlet, which is controlled by a mutex lock. Further, every scaling operation is followed by a stabilization period where no scaling operation takes place and system does not enter the monitoring phase for the next MAPE cycle. The objective of these constraints is to avoid oscillations in scaling activities; for instance, repetitively scaling out in the presence of memory pressure could result in overprovisioning, which would then lead to recurring scale-in operations.

Figure 5a depicts the phases of the scale-out protocol with respect to our example in Figure 2 when sketchlet C is scaling out to sketchlet D. Once the sketchlet decides on subregions to scale, it initiates the scale-out protocol by contacting the *deployer* process, which is responsible for launching tasks. In this message, it includes a list of preferred sketchlets for the load migration as well as memory requirements and expected message rate for the load. The preferred sketchlet set includes the sketchlets that already hold other subregions. It minimizes the number of sketchlets responsible for each geographical region to reduce communication during query evaluations.

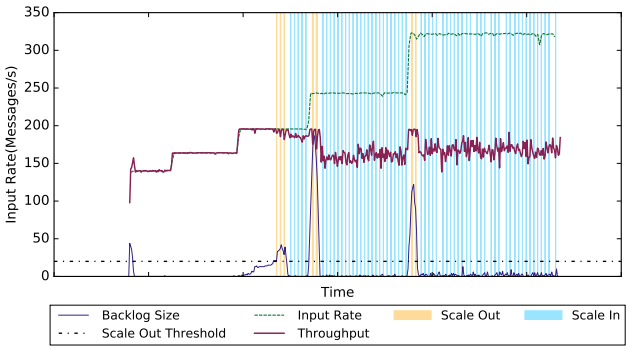
The SYNOPSIS deployer component has an approximate view of the entire system gathered through gossip messages, which includes the memory pressure and cumulative backlog information for each sketchlet. Based on this view and the information present in the request, the deployer replies back with a set of candidate target sketchlets. Only if a suitable candidate cannot be found from the set of current sketchlets will a new sketchlet be spawned. Upon receiving a response from the deployer, the sketchlet (parent) contacts the target sketchlet (child) and tries to acquire the mutex. A lock will be granted only if the target can accommodate the load and no other scaling operations are taking place. If the lock acquisition fails, another candidate from the list is attempted; otherwise, the parent sketchlet will create a pass-through channel and direct traffic corresponding to migrated regions towards the child sketchlet. Once this process is complete, the parent sketchlet will initiate a state transfer asynchronously using a background channel to ensure the stream data flow is not affected, and update child sketchlet's memory utilization metrics to account for the pending state transfer.

As the data flow tree grows with scaling out operations, having parent sketchlets pass traffic through to their children becomes inefficient because of higher bandwidth consumption as well as increased latency due to additional network hops the packets have to travel through. To circumvent this, we allow *short circuiting*, which directs traffic from stream ingesters straight to child sketchlets. For instance, stream ingesters will send data directly to sketchlet D using the short circuited route bypassing sketchlets A and C in Figure 2. We use our gossiping subsystem to update parent sketchlets on child's performance metrics, which is useful for scaling in as explained in Section 3.5.

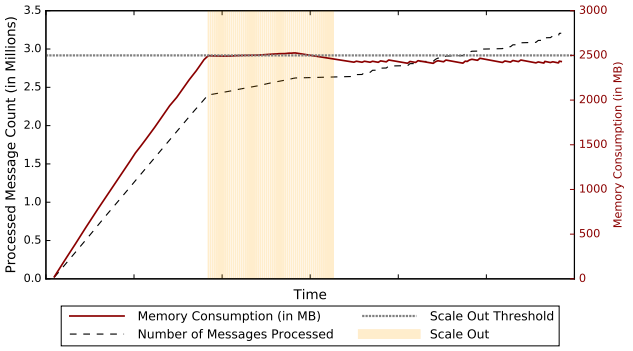
We captured how backlog length and throughput at an individual sketchlet vary with the input rate when dynamic scaling is enabled. The sketchlet that was considered for the experiment immediately received data from stream ingesters, hence the input rate observed at the sketchlet closely resembled the varying data ingestion rate. As shown in Figure 6a, scaling out helps a sketchlet to keep up with the variations in the workload which in turn causes the backlog to stay within a safe range. It also demonstrates infrequent, rapid scale-out and continuous, gradual scale-in as explained in §3.4. Figure 6b demonstrates how memory consumption based threshold-based rules trigger scaling maneuvers to maintain the stability of an individual sketchlet. For this experiment, we have enabled only a single threshold-based rule (either backlog growth based or memory usage based) at a time to demonstrate its effectiveness. We have used a 0.45 of the total available memory for a JVM process as the upper threshold for triggering a scale-out operation. In certain occasions, it is required to perform multiple consecutive scaling out operations (interleaved with the cooling down periods) to bring the memory usage to the desired level due to the increased memory utilization caused by the data ingestion happening in the background.

3.5 Scaling In: Conserving Resources

During scaling in, sketchlets merge back some of the subregions scaled out previously. This ensures better resource utilization in the system in addition to efficient query evaluations by having to contact fewer sketchlets. Scaling in is also guarded by the same mutex lock used for scaling out (only one scale-out or scale-in operation takes place at a given time) and is also followed by a stabilization period.



(a) Dynamic scaling maneuvers triggered by backlog growth based threshold rules



(b) Dynamic scaling maneuvers triggered by memory usage based threshold rules

Fig. 6: Scaling out enables maintaining stability at an individual sketchlets based on backlog growth and memory usage

Monitoring and analysis during scale-in operations proceeds similarly to scaling out, except for the obvious change to the threshold-based rules: now both memory pressure and backlog length metrics should consistently record values below a predefined lower threshold. When scaling in, we use a less aggressive scheme than scaling out; a single subregion is acquired during a single scale-in operation. Scaling in is more complex than scaling out because it involves more than one sketchlet in most cases. At this point, it is possible that further scale-out operations have taken place in the scaled out subregion after the initial scale-out. For instance, if sketchlet A in Figure 2 decides to scale-in the subregion *DN*, then it must communicate with sketchlets C and D.

The scale-in protocol starts with a lock acquisition protocol similar to scaling out protocol, but locking the entire subtree is required. The steps are depicted in Figure 5b with respect to our example in Figure 2 where sketchlet C is scaled in. As per our example, sketchlet A will have to acquire locks for both sketchlets C and E. Locks are acquired in a top-to-bottom fashion where parent locks itself and then attempts to lock the child. If lock acquisition is failed in any part of the subtree, then the scale-in operation is aborted and the monitoring process will start the next iteration of the MAPE loop immediately. If the subtree is successfully locked, then data flow to the child sketchlet corresponding to this subregion is immediately terminated.

The state acquisition phase begins next. To ensure that SYNOPSIS does not lose any messages, the initiating sketchlet sends a *termination point* control message to the child sketchlet. The termination point is the sequence number of the last message sent to the child sketchlet either by the parent itself or by the short circuit channel. Once the child sketchlet has processed every message up to the termination point, it sends out termination point messages to all relevant child sketchlets. In our example, sketchlet C sends a termination point control message to D upon processing the stream packet corresponding to the termination point sent by sketchlet A. After the entire subtree has seen all messages up to the termination point, they acknowledge the initiator sketchlet and start transferring their states asynchronously. Once the parent sketchlet receives acknowledgments from the entire subtree, it starts propagating the protocol end messages to initiate lock releasing. Locks are released from the bottom to top in the subtree, with the parent sketchlet releasing its lock after each child has released its lock.

3.6 Query Evaluations

SYNOPSIS incorporates support for user-defined queries that are evaluated over the distributed sketch. Queries can be specified by the user in a SQL-like format or with JSON-based key-value descriptions similar to GraphQL [17]. Exact-match, range-based, and summarization queries are all supported over spatiotemporal bounds and individual feature values. The following example depicts how SQL-like queries can be formulated for evaluation over the sketch.

```
SELECT MEAN(precipitation), MAX(wind_speed)
WHERE temperature > 20 AND temperature < 30
AND humidity > .8 AND CORRELATION(
    cloud_cover, precipitation) < -0.25
```

Depending on scaling operations and the spatial scope of the queries, evaluations are carried out on one or more sketchlets. Information on the placement of sketchlets in the system and their corresponding feature scopes is maintained at each sketchlet in a geohash prefix tree, with changes propagated through the network in an eventually-consistent manner as data is ingested and scaling maneuvers occur.

The entry point for these queries, called the *conduit*, may be any of the sketchlets comprising the distributed sketch. During query evaluations, the first step is to identify the set of sketchlets that are relevant to the query. The conduit consults its prefix tree to locate sketchlets based on spatial, chronological, and feature constraints specified by the user. After this process is complete, the conduit forwards the queries on to the sketchlets for evaluation and supplies the client with a list of sketchlets that will respond to the query. As queries execute, results are streamed back to the client and merged by the client API. This strategy ensures that I/O and processing activities are interleaved on the client side.

Our distributed prefix tree enables query evaluations during both scaling in and out. For instance, when a conduit attempts to forward a query to a child sketchlet that is undergoing a scale-in operation, the request will be redirected to the its parent sketchlet. This process can continue recursively up through the network, ensuring queries will reach their destination.

3.6.1 Query Types

SYNOPSIS queries can be discrete or continuous. Unlike discrete queries that are evaluated once, continuous queries are evaluated periodically or when observations become available. Queries supported by SYNOPSIS fall into six categories:

Relational Queries describe the feature space in the context of the hierarchical trees within our SIFT data structure and may target ranges of values under certain conditions. For example, "What is the relationship between temperature and humidity during July in Alaska, USA, when precipitation was greater than 1 centimeter?" These queries return a subset of the overall sketch that includes other matching feature values as well.

Statistical Queries allow users to explore statistical properties of the observational space by retrieving portions of the metadata stored in the leaf nodes of the sketch. For example, users can retrieve and contrast correlations between any two features at different geographic locations at the same time. Alternatively, queries may contrast correlations between different features at different time ranges at the same geographic location. These queries also support retrieval of the mean, standard deviation, and feature outliers based on Chebyshev's inequality [18].

Density Queries support analysis of the distribution of values associated with a feature over a particular spatiotemporal scope. These include kernel density estimations, estimating the probability of observing a particular value for an observation, and determining the deciles and quartiles for the observed feature.

Set Queries target identification of whether a particular combination of feature values was observed, estimating the cardinality of the dataset, and identifying the frequencies of the observations. Each type of set query requires a particular data structure, with instances created across configurable time bounds (for instance, every day). Set membership is determined using space-efficient bloom filters [19], while cardinality (number of distinct elements) queries are supported by the HyperLogLog [20] algorithm.

Inferential Queries Inferential queries enable spatiotemporal forecasts to be produced for a particular feature (or set of features). Discrete inferential queries leverage existing information in the distributed sketch to make predictions; aggregate metadata stored in the leaves of the tree can produce two-dimensional regression models that forecast new outcomes across each feature type when an independent variable of interest changes.

Synthetic Data Queries allow users to request the system to generate representative datasets based on the distributions stored in the sketch. Synthetic data is generated in an online fashion by sampling from the underlying distributions and then streamed to client applications for analytics. The size of the dataset may also be specified; for instance, 10% of the volume of the original data points.

Table 2 outlines the local sketchlet traversal times for both evaluation types on a single sketchlet, and also includes results that combine the two types (retrieving a tree that also includes metadata). In general, conventional lookups require less processing time. While tree lookups take longer to complete, it is worth noting that varying the geographical scope across sketchlet sizes from 5km to 800km did not result in a proportionate increase in processing time. Overall, the sketch is able to satisfy our goal of low-latency query evaluations for each query type.

3.7 Coping with Failures in Synopsis

SYNOPSIS relies on passive replication to recover from sketchlet failures. Other techniques such as active replication increase the resource consumption significantly and it is infeasible to use upstream backups because the state of a Sketchlet depends on the entire set of stream packets it has processed previously [21].

Support for fault tolerance is implemented by augmenting the distributed sketch) with a set of secondary sketchlets. Each sketchlet is assigned with a set of n secondary sketchlets each running on different machines, so that Synopsis can withstand up to n concurrent machine failures. In our implementation, we used two secondary sketchlets assigned to each sketchlet. The primary sketchlet periodically sends the changes to its in-memory state to the secondary nodes creating an stream of edits to the sketchlet between the primary and secondary sketchlets. This incremental check-pointing scheme consumes a less bandwidth compared to a periodic check-pointing scheme that replicates the entire state every time [21]. The secondary sketchlets, which acts as the sink to the edit stream serializes the messages received via the edit stream to persistent storage. By default, SYNOPSIS uses the disk of the machine which hosts the secondary sketchlet for persistent storage, but necessary API level provisions are included to support highly available storage implementations such as HDFS [7]. To reduce the overhead caused by secondary sketchlets, they do not load this serialized state into its memory unless there is a failure to the primary and consequently it gets appointed as the primary sketchlet using the leader election protocol implemented using Zookeeper [22].

Incremental checkpoints are performed based on a special control message emitted by the stream ingesters. These messages help to orchestrate a system-wide incremental checkpoint. SYNOPSIS uses upstream backups at stream ingesters to keep a copy of the messages that entered the system since the last successful checkpoint. In case of a failure, all messages since the last checkpoint will be replayed.

A sketchlet is implemented as an idempotent data structure using message sequence numbers, hence it will process a replayed message only if the message is not processed before. Users can apply their own policy for defining the interval between the incremental checkpoints based on time or the number of emitted messages. Frequent checkpoints can incur high overhead whereas longer periods between successive checkpoints may consume more resources for upstream backups as well as for replaying messages in case of a failure.

Membership management is implemented using Zookeeper, which is leveraged to detect failed nodes. Upon receiving node failure notifications, a secondary sketchlet is assumed the role of the primary. The secondary will start processing messages immediately and start populating its state from the persistent storage in the background. Given this mode of operation, there may be a small window of time during which the correctness of queries are impacted. This is rectified once the stored state is loaded to memory and replay of the upstream backup is completed. The sketch's ability to correctly process out of order

TABLE 2: Local sketchlet evaluation times for each query type (averaged over 1000 iterations).

Query Type	Eval. (ms)	Std. Dev.
Density	0.007	0.005
Set Cardinality	0.154	0.088
Set Frequency	0.036	0.019
Set Membership	0.015	0.009
Statistical	0.002	0.001
Tree Only (5 km)	46.357	1.287
Tree + Meta (5 km)	40.510	6.937
Tree + Meta (25 km)	47.619	6.355
Tree + Meta (800 km)	53.620	6.818

messages and support for merging with other sketches is useful during this failure recovery process.

As per our fault tolerance model, the *total time to recover from the failure* (T_{total}) can be modeled by the following equation.

$$T_{total} = T_d + \max(T_l, T_r)$$

where T_d = *time to detect a failure*, T_l = *time to load persisted state* and T_r = *replay time for messages at the upstream node*.

Time to detect failure mainly depends on the session timeout value used by the Zookeeper to detect lost nodes and the delay in propagating the notification about the lost cluster-node to other members. With a $5s$ session timeout in an active cluster, we observed a mean notification propagation delay of $5.500s$ (std. deviation = $0.911s$, 95^{th} Percentile = $6.000s$). Configuring a lower session timeout will increase the chance of false negatives if cluster-nodes become non responsive for a while due to increased load or system activities such as garbage collection. Using a higher session timeout will increase the time to detect failures. Time required to load the persisted storage depends on the size of the serialized sketchlet. We benchmarked the time it takes to repopulate the state in all sketchlets after ingesting NOAA data for year 2014. The mean state re-population time was recorded as $16.602s$ with a std. deviation of $23.215s$ and a 95^{th} percentile of $70.877s$. Replay time mainly depends on the check-pointing interval as well as the message ingestion rate. With a check-pointing interval of 10000 messages, we experienced a mean value of $0.447s$ (std. deviation = $0.036s$, 95^{th} Percentile = $0.484s$) to replay the buffered messages from stream ingesters.

4 Performance Evaluation

Here we report system benchmarks profiling several aspects of Synopsis.

4.1 Dataset and Experimental Setup

Our subject dataset was sourced from the NOAA North American Mesoscale (NAM) Forecast System [10]. The NAM collects atmospheric data several times per day and includes features of interest such as surface temperature, visibility, relative humidity, snow, and precipitation. Each observation in the dataset also incorporates a relevant geographical location and time of creation. This information is used during the data ingest process to partition streams across available computing resources and preserve temporal ordering of events. The size of the entire source dataset was 25 TB.

Performance evaluations reported here were carried out on a cluster of 40 HP DL160 servers (Xeon E5620, 12 GB RAM). The test cluster was configured to run Fedora 24, and Synopsis was executed under the OpenJDK Java runtime 1.8.0_72.

4.2 Distributed Sketch Memory Evaluation

We monitored the growth in memory consumption of the entire distributed sketch over time with continuous data ingestion as shown in Figure 7. As more data was streamed into the system, the growth rate of the distributed sketch decreased as the sketchlets expanded to include vertices for their particular feature space. At the end of our monitoring period, the total amount of ingested data was over three orders of magnitudes higher ($\sim$ 1285) than the in-memory sketch size, resulting in notable space savings.

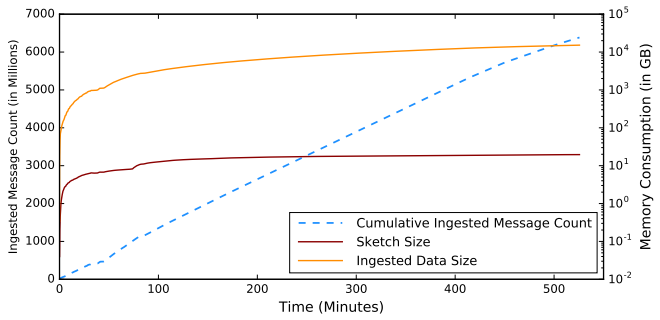


Fig. 7: Memory usage of the distributed sketch over time against the amount of ingested data. The rate of growth decreases over time due to the compact nature of sketchlet data structure.

4.3 Dynamic Scaling: Responding to Variable Load

We evaluated how Synopsis dynamically scales when the data ingestion rate is varied. The data ingestion rate was varied over time such that the peak data ingestion rate is higher than the highest possible cumulative throughput that will create a backlog at sketchlets. We used the number of sketchlets created in the system to quantify the scaling activities. If the system scales out, more sketchlets will be created as a result of targeted load migration. We started with a single sketchlet and allowed the system to dynamically scale. As can be observed in Figure 8, the number of sketchlets varies with the ingestion rate. Since we allow aggressive scale-out, it shows rapid scaling activity during high data ingestion rates whereas scaling in takes place gradually with one sub region (hence one sketchlet) at a time.

4.4 Analyzing a Snapshot of the Distributed Sketch

Figure 9 visualizes a snapshot of the distributed sketch which demonstrates the organization of sketchlets in runtime as described in Section 3. This represents the state of the system after consuming the complete NOAA dataset for 2014 and the graph contained 48 sketchlets. It shows the distribution and size of the information maintained across sketchlets for each geohash prefix of length 3 against the number of records processed for that particular prefix. The memory requirement for a particular geohash prefix depends on the number of records as well as the range of the observed values for different features. The space requirement is measured in terms of the number of leaf nodes in the corresponding sketchlets. For the majority of the prefixes, the space requirement increases with the number of records

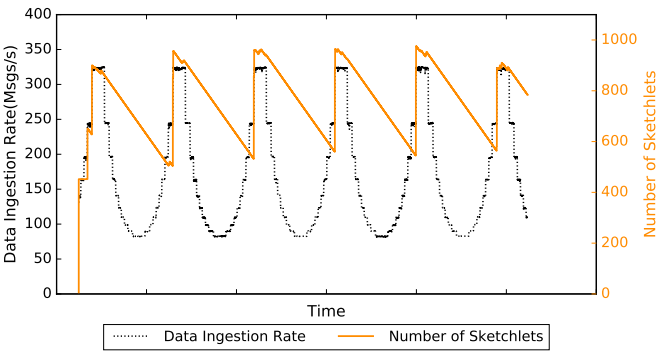


Fig. 8: Variation in the number of sketchlets as the data ingestion rate changes.

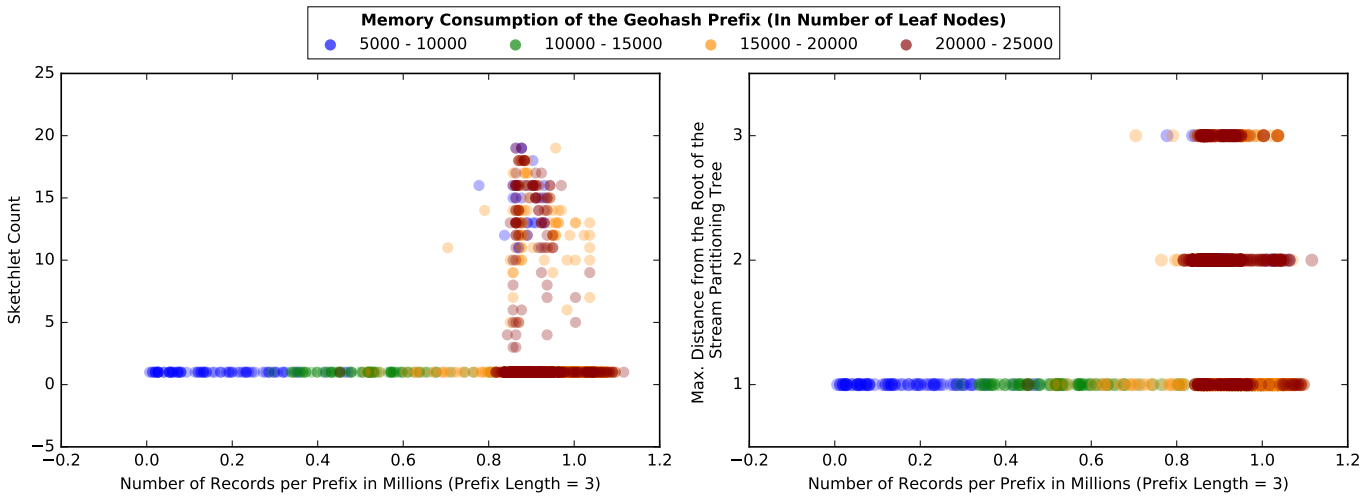


Fig. 9: Analysis of a snapshot of the distributed sketch during data ingestion demonstrating the size and distribution of the information corresponding to different prefixes against the observed record count. If the information is dispersed over multiple sketchlets, it is likely to be a prefix with higher number of records and/or a wide range of observed values.

processed for a particular prefix. If the data for a particular prefix is distributed across multiple sketchlets, then it is more likely to be a prefix with a high number of records as shown in the first subplot. In such cases, some of these sketchlets are created in multiple iterations of scaling out operations from their original sketchlets which results in a higher distance from the root of the prefix tree. This is depicted in the second sub figure of Figure 9. A few prefixes with high number of records can be observed with a low memory consumption and are distributed across multiple sketchlets. Their observations spans across a smaller range, hence requires less memory but they were chosen for scaling out operations due to their high message rates.

4.5 Query Evaluation Performance

To evaluate distributed query performance, we executed several representative workloads across a variety of sketchlet sizes. These queries were separated into two groups: conventional lookups and tree retrievals. Conventional lookups include density queries, set queries, and statistical queries, while tree retrievals request targeted portions of the overall feature space as a tree. Note that while conventional lookups do not return a tree structure to the client, they still require a tree traversal to resolve. In general, tree retrievals consume more processing time due to their serialization and I/O costs.

Figure 10 demonstrates the end-to-end efficiency of the query evaluations over the distributed sketch. Cumulative query throughput and latencies were measured with varying numbers of concurrent query *funnels*. A query *funnel* continuously generates and dispatches random queries at their maximum possible rate to stress test the system and saturate its capacity; for instance, a query could request summary statistics or feature relationships when the temperature is between 20 to 30 degrees, humidity is above 80%, and the wind is blowing at 16 km/h. These randomized queries fluctuated in both the ranges of values and spatial scope, resulting in high variability in the number of sketchlets required to resolve the query as well as the depth and breadth of the required tree traversals.

5 Applications

To demonstrate the effectiveness of Synopsis as a surrogate for the data, we report on its use in two settings normally under the purview of on disk data: visualization and input to analytical engines.

5.1 Visualization

To demonstrate the potential applications of Synopsis, we created two visualizations. Our first visualization generated a climate chart by issuing statistical queries to retrieve high, low, and mean temperature values as well as precipitation information for a given spatial region. Climate charts are often used to provide a quick overview of the weather for a location; Figure 11 summarizes the temperature and precipitation in Snowmass Village, Colorado during 2014. While a standard approach for producing these visualizations over voluminous atmospheric data would likely involve several MapReduce computations, our sketchlets make all the necessary information readily available through queries, avoiding distributed computations altogether. Furthermore, retrieving the data for this evaluation consumed considerably less time (1.5 ms) than rendering the image on the client side (127.1 ms).

Our second visualization issued queries to retrieve cloud cover information for the entirety of North America. To reduce processing load on the client side, we specified minimum visibility thresholds to eliminate data points that would not be visible in the final output figure. After retrieving this information, we executed a second query that located all areas that exhibited high correlations between cloud cover and precipitation. Figure 12 illustrates the results of this process for data in July of 2014; cloud cover is represented by white contours with varying opacity, while blue contours describe the correlation between cloud cover and precipitation (darker blues, such as those seen in the top-center of the globe, represent a stronger correlation). Due to the large scope of this visualization (retrieving all data points for a given month across all spatial regions), retrieval took approximately 2.82 seconds, with graphics rendering consuming an additional 1.51 seconds at the client application.

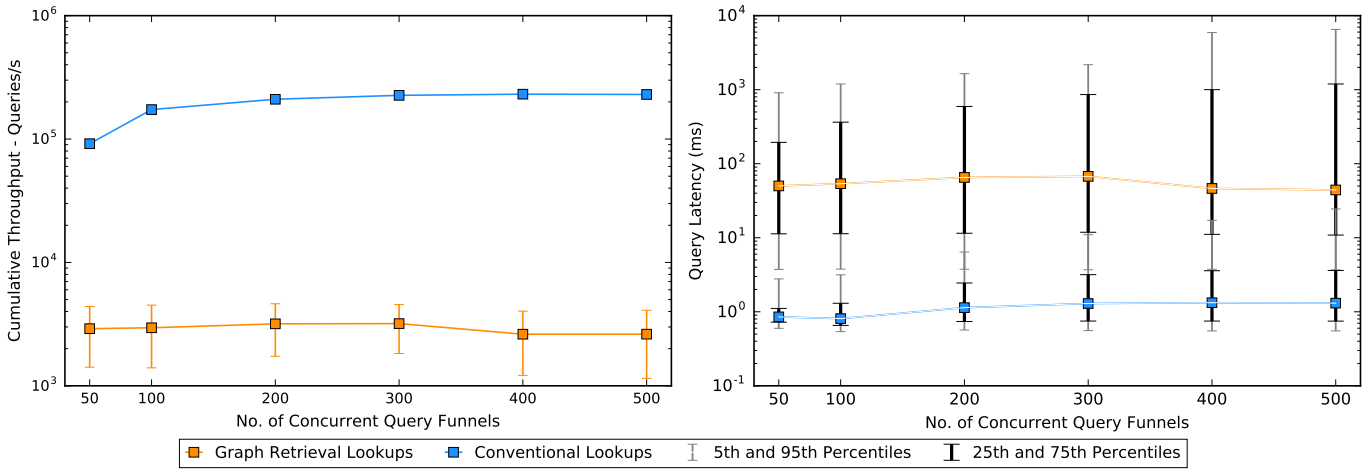


Fig. 10: Distributed Query Evaluation Performance. Variation of cumulative throughput and latency against the number of concurrent query funnels in a 40 node SYNOPSIS cluster.

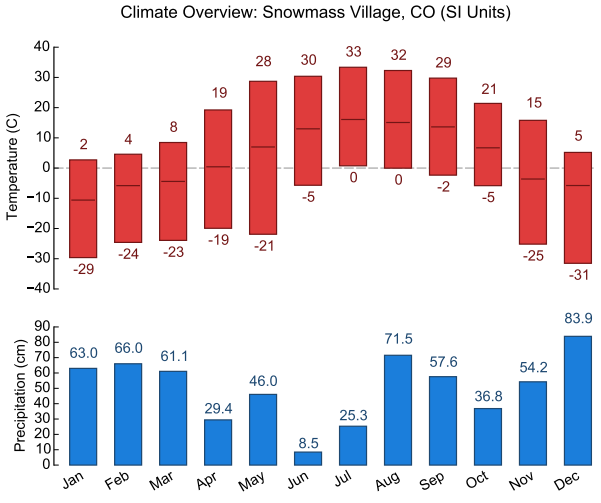


Fig. 11: A climate chart generated with our statistical query functionality.

5.2 Use with Analytic Engines

Synthetic data queries in SYNOPSIS can be used to generate representative datasets that require less space while still providing high accuracy. Such datasets can be used efficiently with analytic engines such as Apache Spark [1] and TensorFlow [4]. We used Spark to train regression models based on the Random Forest ensemble method to predict temperatures using surface visibility, humidity and precipitation in the southeast United States during the months of May–August, 2011–2014. These models were generated using the original full-resolution data as well as synthetic data sets that were sized at 10% and 20% of the original data. The accuracy of these models were measured using a test dataset extracted from actual observations (30% of the overall dataset size). All three datasets were staged on HDFS and loaded into Spark to train the ensemble models. For this benchmark, we used Apache Spark version 2.0.1 with HDFS 2.6.0 with a 100 node cluster by combining our baseline cluster of 40 machines with 30 HP DL320e servers (Xeon E3-1220 V2, 8 GB RAM) and 30 HP DL60 servers (Xeon E5-2620, 16 GB RAM).

We evaluated the efficacy of our approach based on the on-disk and in-memory storage requirements, data loading time, training time and the accuracy of the model. Our observations are summarized in Table 3. The accuracy of the models generated based on synthetic data is comparable to the accuracy of the models generated using actual data, while requiring less space and training time. For instance, our 10% synthetic dataset produces a model with similar accuracy incurring 53% less training time while reducing space requirements by 90%. Additionally, based on the number of RDD partitions used, the synthetic datasets require substantially less computing resources.

6 Related Work

Data Cubes [23], [24], [25], [26] are a data structure for Online Analytical Processing that provide multidimensional query and summarization functionality. These structures generalize several operators provided by relational databases by projecting two-dimensional relational tables to N-dimensional cubes (also known as *hypercubes* when $N > 3$). Variable precision in Data Cubes is managed by the *drill down/drill up* operators to increase and decrease resolution, respectively, and *slices* or entire cubes can be summarized through the *roll up* operator. While Data Cubes provide many of the same features supported by our distributed sketch, they are primarily intended for single-host offline or batch processing systems due to their compute- and data-intensive updates. In fact, many production deployments separate transaction processing and analytical processing systems, with updates pushed to the Data Cubes on a periodic basis.

CHAOS [27] builds on Data Cubes in a single-host streaming environment by pushing updates to its *Computational Cubes* more frequently. To make dimensionality and storage requirements manageable, Computational Cubes only index summaries of incoming data that are generated during a preprocessing step. However, full-resolution data is still made available through continuous queries that act on variable-length sliding windows. CHAOS builds its summaries using a wavelet-based approach, which tend to be highly problem-specific. Additionally, updates to the cube structure are still generated and published on a periodic basis rather than immediately as data is assimilated.

Galileo [28], [29] is a distributed hash table that supports the storage and retrieval of multidimensional data. Given the

TABLE 3: Comparing Random Forest based regression models generated by Spark MLlib using synthetic vs. real data

Dataset	Size (GB)	RDD Partitions	Data Loading Time (s)		Model Training Time (s)		Accuracy (RMSE)	
			Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Original	25.350	208	4.611	0.133	520.657	7.690	6.025	0.051
Synthetic - 10%	2.549	21	3.424	0.288	246.212	15.046	5.980	0.024
Synthetic - 20%	4.336	41	3.997	0.392	278.682	17.475	6.018	0.064

overlap in problem domain, Galileo is faced with several of the same challenges as SYNOPSIS. However, the avenues for overcoming these issues diverge significantly due to differences in storage philosophy: SYNOPSIS maintains its dataset completely in main memory, avoiding the orders-of-magnitude disparity in I/O throughput associated with secondary storage systems. This makes SYNOPSIS highly agile, allowing on-demand scaling to rapidly respond to changes in incoming load. Additionally, this constraint influenced the trade-off space involved when designing our algorithms, making careful and efficient memory management a priority while striving for high accuracy.

Simba (Spatial In-Memory Big data Analytics) [30] extends Spark SQL [2] to support spatial operations in SQL as well as DataFrames. It relies on data being stored in Spark [1]. Even though this approach provides a high accuracy, it is not scalable for geospatial streams in the long term due to high storage requirements. In SYNOPSIS, spatial queries can be executed with a reasonable accuracy without having to store the streaming data as-is.

Dynamic scaling and elasticity in stream processing systems has been studied thoroughly [31], [32], [21], [33], [34], [35]. Heinze et al. [31] explores using different dynamic scaling schemes including threshold-based rules and reinforcement learning using the FUGU [34] stream processing engine. Based on these schemes, the operators are continuously migrated between hosts in a FUGU cluster in order to optimize the resource utilization and to maintain low latency. Their approach is quite different from ours, because in SYNOPSIS we perform a targeted load migration where the workload of a computation is dynamically adjusted instead of entirely moving it to a host with a higher or lower capacity than the current host. Further, we do not interrupt the data flow through SYNOPSIS when dynamic scaling activities are in progress, whereas in FUGU the predecessor operator is temporarily paused until operator migration is complete.

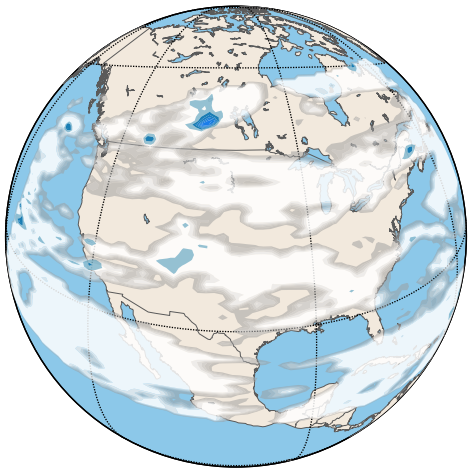


Fig. 12: Global contour visualization showing cloud cover (white contours) and the correlation with precipitation (blue contours) in July of 2014 across North America.

StreamCloud [32] relies on a global threshold-based scheme to implement elasticity where a query is partitioned into subqueries which run on separate clusters. StreamCloud relies on a centralized component, the Elastic Manager, to initiate the elastic reconfiguration protocol, whereas in Synopsis each node independently initiates the dynamic scaling protocol. This difference is mainly due to different optimization objectives of the two systems; StreamCloud tries to optimize the average CPU usage per cluster while SYNOPSIS attempts to maintain stability at each node. The state recreation protocol of StreamCloud is conceptually similar to our state transfer protocol, except in StreamCloud the tuples are buffered at the new location until the state transfer is complete, whereas in SYNOPSIS the new sketchlet immediately starts building the state which is later merged with the asynchronously transferred state from the parent sketchlet.

Gedik et al. [35] also uses a threshold-based local scheme similar to SYNOPSIS. Additionally, this approach keeps track of the past performance achieved at different operating conditions in order to avoid oscillations in scaling activities. The use of consistent hashing at the splitters (similar to geohash based stream partitioning in SYNOPSIS) achieves both load balancing and monotonicity (elastic scaling does not move states between nodes that are present before and after the scaling activity). Similarly, our geohash-based partitioner together with control algorithms in SYNOPSIS balance the workload by alleviating hotspots and cluster-nodes with lower resource utilization. Our state migration scheme doesn't require migrating states between sketchlets that do not participate in the scaling activity, unlike with a reconfiguration of a regular hash-based partitioner. Unlike in SYNOPSIS, in their implementation, the stream data flow is paused until state migration is complete using vertical and horizontal barriers. Finally, SYNOPSIS' scaling schemes are placement-aware, meaning certain nodes are preferred when performing scaling with the objective of reducing the span of the distributed sketch.

7 Conclusions and Future Work

Our framework for constructing a distributed sketch over spatiotemporal streams, SYNOPSIS: (1) maintains a compact representation of the observational space, (2) supports dynamic scaling to preserve responsiveness and avoid overprovisioning, and (3) supports a rich set of queries to explore the observational space. Our methodology for achieving this is broadly applicable to other stream processing systems. Our empirical benchmarks, with real-world observational data, demonstrate the suitability of our approach.

We achieve compactness in our sketchlet instances by dynamically managing the number of vertices in the SIFT hierarchy as well as the size of the ranges each vertex is responsible for. We also maintain summary statistics and metadata within these vertices to track the distribution/dispersion of feature values and their frequencies. As a result, SYNOPSIS is able to represent datasets using substantially less memory (**RQ-1**). Given variability in the rates and volumes of data arrivals from

different geolocations, our scaling mechanism avoids overprovisioning and alleviates situations where sketch updates cannot keep pace with data arrival rates. Memory pressure is also taken into account during replica creation as well as when caling in and out (**RQ-2**). During query evaluations, only the sketchlets that hold portions of the observational space implicitly or explicitly targeted by the query are involved, ensuring high throughput. We support several high-level query operations that allow users to locate and manipulate data efficiently (**RQ-3**).

Our future work will target support for SYNOPSIS to be used as input for long-running computations. Such jobs would execute periodically on a varying number of machines and could target the entire observational space or only the most recently-assimilated records.

ACKNOWLEDGMENTS

This research was supported by the US Dept of Homeland Security [HSHQDC-13-C-B0018, D15PC00279]; the US National Science Foundation [ACI-1553685, CNS-1253908]; and the Environmental Defense Fund [0164-000000-10410-100].

REFERENCES

[1] M. Zaharia, M. Chowdhury *et al.*, "Spark: cluster computing with working sets." *HotCloud*, vol. 10, pp. 10–10, 2010.
[2] M. Armbrust, R. S. Xin *et al.*, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1383–1394.
[3] M. Abadi, "Tensorflow: Learning functions at scale," in *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP 2016. New York, NY, USA: ACM, 2016, pp. 1–1. [Online]. Available: http://doi.acm.org/10.1145/2951913.2976746
[4] M. Abadi, A. Agarwal *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2015. [Online]. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf
[5] The Apache Software Foundation, "Hadoop," *http://hadoop.apache.org*, 2016.
[6] K. Shvachko, H. Kuang *et al.*, "The hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, May 2010, pp. 1–10.
[7] D. Borthakur, "Hdfs architecture guide," *Hadoop Project*, 2008.
[8] J. Langford, L. Li *et al.*, "Vowpal wabbit online learning project," 2007.
[9] G. Niemeyer. (2008) Geohash. [Online]. Available: http://en.wikipedia.org/wiki/Geohash
[10] National Oceanic and Atmospheric Administration. (2016) The North American Mesoscale Forecast System. [Online]. Available: http://www.emc.ncep.noaa.gov/index.php?branch=NAM
[11] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4(3), pp. 419–420, 1962.
[12] M. Kristan, A. Leonardis *et al.*, "Multivariate online kernel density estimation with gaussian kernels," *Pattern Recognition*, vol. 44, no. 1011, pp. 2630 – 2642, 2011, semi-Supervised Learning for Visual Content Analysis and Understanding. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320311001233
[13] M. Malensek, S. Pallickara *et al.*, "Autonomously improving query evaluations over multidimensional data in distributed hash tables," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference (CAC)*, Sep 2013, pp. 15:1–15:10.
[14] B. W. Silverman, *Density estimation for statistics and data analysis*. CRC press, 1986, vol. 26.
[15] M. Maurer, I. Breskovic *et al.*, "Revealing the MAPE loop for the autonomic management of cloud infrastructures," in *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pp. 147–152.
[16] T. Lorido-Botrán, J. Miguel-Alonso *et al.*, "Auto-scaling techniques for elastic applications in cloud environments," *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, vol. 12, p. 2012, 2012.
[17] Facebook, Inc. (2015) GraphQL. [Online]. Available: graphql.org
[18] D. E. Knuth, "The art of computer programming: Fundamental algorithms (volume 1)," *Addison-Wesley*, 1968.
[19] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13(7), pp. 422–426, 1970.
[20] P. Flajolet, É. Fusy *et al.*, "HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm," in *AOFA '07: Proceedings of the 2007 International Conference on the Analysis of Algorithms*.
[21] R. Castro Fernandez, M. Migliavacca *et al.*, "Integrating scale out and fault tolerance in stream processing using operator state management," in *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. ACM, 2013, pp. 725–736.
[22] P. Hunt, M. Konar *et al.*, "Zookeeper: Wait-free coordination for internet-scale systems." in *USENIX Annual Technical Conference*, vol. 8, 2010, p. 9.
[23] J. Gray, A. Bosworth *et al.*, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total," in *Proceedings of the Twelfth International Conference on Data Engineering*, ser. ICDE '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 152–159. [Online]. Available: http://dl.acm.org/citation.cfm?id=645481.655593
[24] V. Harinarayan, A. Rajaraman *et al.*, "Implementing data cubes efficiently," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '96. New York, NY, USA: ACM, 1996, pp. 205–216. [Online]. Available: http://doi.acm.org/10.1145/233269.233333
[25] I. S. Mumick, D. Quass *et al.*, "Maintenance of data cubes and summary tables in a warehouse," in *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '97, New York, NY, USA, pp. 100–111.
[26] C.-T. Ho, R. Agrawal *et al.*, "Range queries in olap data cubes," in *Proc. of the 1997 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, pp. 73–88.
[27] C. Gupta, S. Wang *et al.*, "Chaos: A data stream analysis architecture for enterprise applications," in *2009 IEEE Conference on Commerce and Enterprise Computing*, July 2009, pp. 33–40.
[28] M. Malensek, S. Pallickara *et al.*, "Analytic queries over geospatial time-series data using distributed hash tables," *IEEE Transactions on Knowledge and Data Engineering*, vol. PP, no. 99, pp. 1–1, 2016.
[29] ——, "Fast, ad hoc query evaluations over multidimensional geospatial datasets," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2015.
[30] D. Xie, F. Li *et al.*, "Simba: Efficient in-memory spatial analytics."
[31] T. Heinze, V. Pappalardo *et al.*, "Auto-scaling techniques for elastic data stream processing," in *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on*. IEEE, 2014, pp. 296–302.
[32] V. Gulisano, R. Jimenez-Peris *et al.*, "Streamcloud: An elastic and scalable data streaming system," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 12, pp. 2351–2365, 2012.
[33] S. Loesing, M. Hentschel *et al.*, "Stormy: an elastic and highly available streaming service in the cloud," in *Proceedings of the 2012 Joint EDBT/ICDT Workshops*. ACM, 2012, pp. 55–60.
[34] T. Heinze, Y. Ji *et al.*, "Elastic complex event processing under varying query load." in *BD3@ VLDB*. Citeseer, 2013, pp. 25–30.
[35] S. Schneider, H. Andrade *et al.*, "Elastic scaling of data parallel operators in stream processing," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–12.

Thilina Buddhika Thilina Buddhika is a Ph.D. candidate in the Computer Science department at Colorado State University. His research interests are in the area of real time, high throughput stream processing specifically targeted to environments such as Internet of Things (IoT) and health care applications.

Matthew Malensek Matthew Malensek is a Ph.D. student in the Department of Computer Science at Colorado State University. His research involves the design and implementation of large-scale distributed systems, data-intensive computing, and cloud computing.

Sangmi Lee Pallickara Sangmi Lee Pallickara is an Assistant Professor in the Department of Computer Science at Colorado State University. She received her Masters and Ph.D. degrees in Computer Science from Syracuse University and Florida State University, respectively. Her research interests are in the area of large-scale scientific data management, data mining, scientific metadata, and data-intensive computing.

Shrideep Pallickara Shrideep Pallickara is an Associate Professor in the Department of Computer Science and a Monfort Professor at Colorado State University. His research interests are in the area of large-scale distributed systems, specifically cloud computing and streaming. He received his Masters and Ph.D. degrees from Syracuse University. He is a recipient of an NSF CAREER award.