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

Anonymous
Address Line 1
Address Line 2
City, State/Province, Country
email@address.com

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, a stream processing system that builds an online distributed *sketch* of incoming observational 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.

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 decision making, scientific modeling, simulations, and resource allocation. 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. The queries may be continuous or discrete, involve sliding windows, and encompass varying geospatial scopes.

1.1 Challenges

Support for real-time evaluation of queries — discrete and continuous — 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. Furthermore, this rate of arrivals 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 specified by the user 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 implementing this functionality led us to formulate the following research questions:

SIGMOD '17 May 14–19, 2017, Raleigh, NC, USA
© 2016 ACM. ISBN 978-1-4503-2138-9…$15.00
DOI: 10.1145/1235

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.

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 City would have a far higher rate of observations than Denver, Colorado.

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.

1.3 Approach Summary

Our methodology targets (1) creation of the distributed sketch, (2) updating the sketch in response to data arrivals, (3) splitting and fusing sketch instances to deal with hotspots, (4) using the distributed sketch to perform query evaluations, and (5) achieving low latency by minimizing synchronization between different components in the system.

The sketch supports three key features: maintaining a compact representation of incoming data streams, splitting and fusing portions of the overall sketch, and expressive query evaluations. Relationships between observations and their values are maintained in a graph-based structure that employs several online, in-memory summarization techniques. The sketch is also naturally amenable to distribution, with each node in the cluster holding information about a particular subset of the observational space. This ensures each node in the system can evaluate multiple concurrent queries independently.

Distributing the sketch across multiple 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. Additionally, the sketch updates its precision dynamically to ensure representativeness of the dataset. This is achieved in one of two possible modes: biasing towards the recent past, or ensuring equal representation. In the former approach, the sketch targets finer-grained representations of observations in the recent past, and progressively coarser grained representations for the farther past. This scheme ensures that the apportioning of the available memory for the sketch is proportional to the time intervals under consideration. In the latter approach, the sketch adaptively moves toward a coarser-grained representation as data accumulates.

1.4 Paper Contributions

In this paper, we present a framework for real-time query evaluations over voluminous, time-series data streams. Our specific contributions include:

- Design of a sketch that maintains compact, distributed representations of the observational space with support for low-latency queries.
- Dynamic scaling of the sketch in response to data arrival rates, 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. Most importantly, the framework achieves this while maintaining acceptable levels of accuracy in query evaluations.
- Support for discrete and continuous query evaluations across the observational space. Query evaluations can target arbitrarily sized (chronological) window sizes and geospatial scopes.

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

1.5 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. Section 4 contains our performance evaluation of the system. Section 5 discusses related approaches, and Section 6 concludes the paper and outlines our future research direction.

2. SYSTEM OVERVIEW

SYNOPSIS is a distributed sketch constructed over voluminous data streams. The system is composed of several hosts that form a cluster, with each participating host managing one or more SYNOPSIS *nodes*. Each node contains a *sketchlet*, which maintains multidimensional data points falling within its assigned geospatial scope(s). The number of nodes that comprise the sketch varies dynamically as the sketch scales in or out to cope with data arrival rates and memory pressure. A stream partitioning scheme, based on the Geohash algorithm (described in Section 3.2), is used to route packets to the appropriate sketchlet. Sketchlets ingest stream packets and construct compact, in-memory representations of the observational data by extracting metadata from individual stream packets. During dynamic scaling operations, the geographical extents managed by a sketch varies.

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. These services include the control plane as well as the gossip, querying, and monitoring subsystems.

Control plane: The control plane is responsible for orchestrating control messages exchanged between SYNOPSIS nodes as part of various distributed protocols such 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 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 an 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 distributed evaluation of queries. This involves forwarding 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 rate of packet arrivals and updates to the in-memory structures. This information is used for dynamic scaling recommendations as explained in in Section 3.3.

3. METHODOLOGY

Enabling real-time query evaluations over voluminous spatiotemporal data streams requires several individual components that must coordinate processing and scaling activities over a cluster of heterogeneous resources. This includes our distributed sketch, partitioning scheme, scaling logic, and support for fault tolerance.

3.1 Sketch Construction

Our *sketchlet* data structure is a multidimensional, graph-based model of incoming data streams. Each resource in the system maintains an in-memory sketchlet instance that can be queried to retrieve statistical properties about the underlying data or discover how features interact. Due to the magnitude of inbound data volumes, 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.

3.1.1 Graph Structure

Sketchlet instances are maintained as hierarchical graphs with feature values stored in the vertices. Each *plane* in the graph represents a particular *feature type*, and traversing through vertices in this feature hierarchy reduces the search space of a query. Paths taken through the graph during a lookup operation are influenced by the specificity of the query, with additional feature expressions constraining the *query scope*. Figure 1 demonstrates the structure of a sketchlet and highlights a query and its scope. Note that vertices on the same plane are connected to allow range queries, and that any subset of the graph can be retrieved and manipulated in the same fashion as the overall sketch.

Metadata records for paths through the feature hierarchy are stored at leaf nodes. Each record contains a variety of statistics that are updated in an online fashion using Welford's Algorithm [27]. This includes cross-feature relationships, such as the correlation between temperature values and humidity or the reflectivity of the earth and cloud cover. 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 on disparate resources and then fused into a single, coherent result sketch.

The number of unique feature types stored in the graph directly influences the size of the hierarchy, impacting memory consumption. However, the number of vertices and edges that must be maintained by a graph 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 graph. Most importantly, leaf vertices must contain the spatial locations of the records to facilitate our scaling strategy; storing this information at the top of the hierarchy would simply result in an individual sketchlet being maintained for each spatial location, eliminating the memory consumption benefits of the data structure. Feature planes are reconfigured dynamically based on their corresponding vertex *fan-out* during the initial population of the graph. In this phase full-resolution feature values are stored at each vertex, but once a steady state is reached the *quantization* process begins.

3.1.2 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 sketchlet instances we perform *quantization* — targeted reduction of resolution — which allows vertices in the graph 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 collection of observations over a range of values.

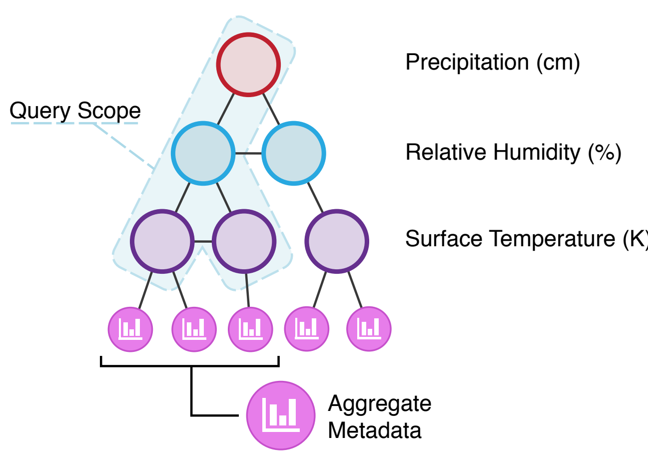


Figure 1: A simplified sketchlet with a three-plane hierarchy and sample query scope, leading to several metadata leaves. In production settings, sketchlets contain hundreds of thousands of vertices and edges.

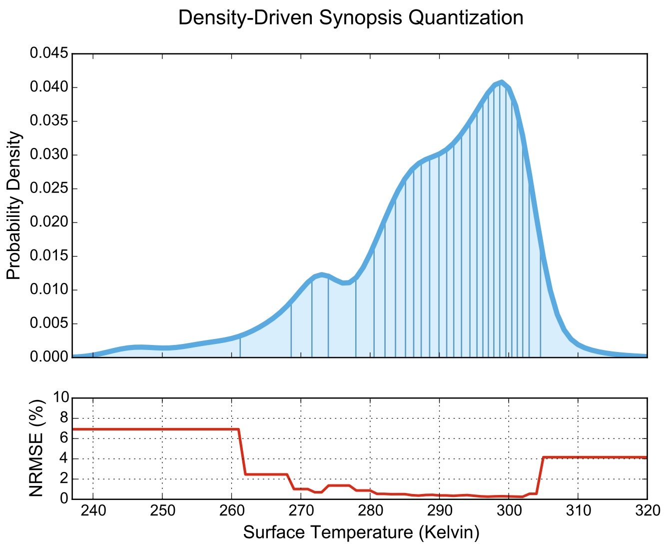


Figure 2: A demonstration of the quantization process, with 29 vertex bins generated across the distribution of surface temperature values in our dataset. Each bin is indicated by a vertical line under the curve.

To determine the size and quantity of these bins, sketchlets maintain additional graph metadata provided by the multivariate online kernel density estimation (oKDE) algorithm developed by Kristan et al. [16]. 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 [26]. Silverman's rule assumes that data tends to follow a normal distribution, which is generally true for naturally-occurring observations such as those in our dataset. 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 2 illustrates the quantization process for the *surface temperature* feature in our atmospheric test dataset [23]: the highest densities of values are stored in the smallest bins (indicated by vertical lines under the curve), improving overall accuracy. 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 graph 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 a full-resolution and quantized sketchlet of 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 sketchlet instance. Decreasing the memory footprint of the sketchlet also allows larger geographical areas to be maintained by a single node.

Table 1: Graph statistics before and after our dynamic quantization algorithm.

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%

3.1.3 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 graph paths from the sketchlet hierarchy in the oldest subgraphs, 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.

3.2 Stream Partitioning

We use the Geohash algorithm [24] to balance load and partition incoming data streams across processing resources. 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 3 illustrates this hierarchy. Most of the eastern United States is contained within the bounding box described by Geohash string 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 resources 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.

Figure 4 depicts a possible arrangement of the distributed sketch and the associated stream partitioning scheme corresponding to our example region. The distributed sketch is arranged in a tree-like structure. Stream ingesters act as root nodes of the tree and partition the stream among Synopsis nodes using a Geohash-based partitioning function. Nodes closer to the root hold sketches corresponding to shorter Geohash strings, and therefore larger geographical regions. For instance, nodes A and B in Figure 4 are responsible for regions represented by Geohashes *DJ* and and *DN* respectively. Node E, which is three edges deep from the stream ingester, is responsible for a smaller region, *DJKJ*.

Synopsis is designed to ensure that regions corresponding to larger portions of the input streams (hence, more frequent updates) are moved to dedicated or less crowded nodes. If a node decides to scale out a portion of the region it is currently responsible for, then the corresponding state (sketchlet and related metadata) is transferred over to the new computation and it starts to treat the stream packets corresponding to the scaled out region as pass-through traffic (Scaling out is explained in section 3.3). More specifically, the node will not process the stream packet, but instead updates its statistics based on the headers of the packet and forwards it to the destination child node. For instance, node A in Figure 4 has scaled out two regions (*DJK* and *DJM*) to

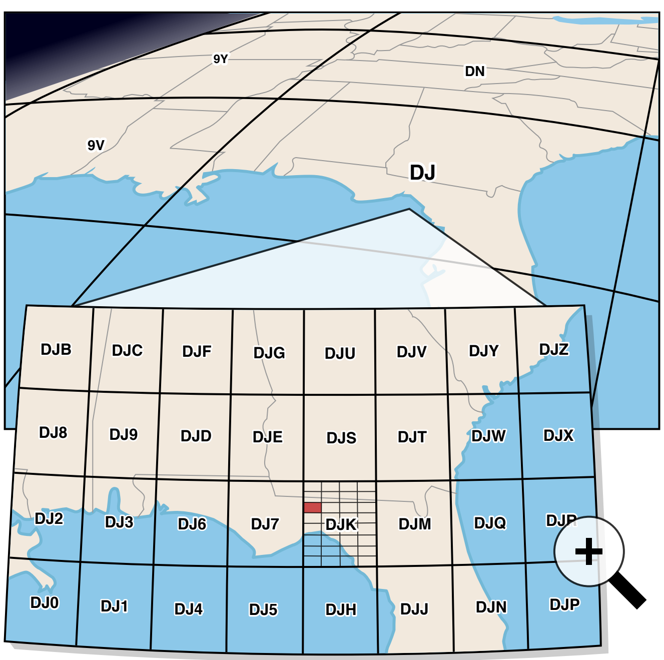


Figure 3: A demonstration of the Geohash algorithm. Each additional character in a Geohash string describes a finer-grained region; Geohash *DJ* contains substantial parts of Florida, Georgia, and Alabama, USA, while *DJKJ* (highlighted in red) encompasses Tallahassee, the capital of Florida.

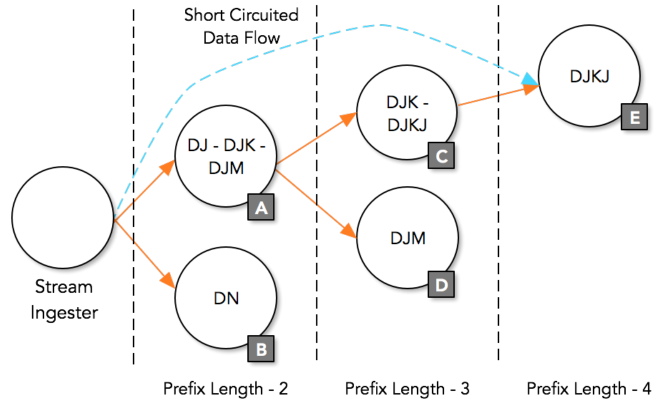


Figure 4: An example of our stream partitioning approach based on the Geohash values of incoming packets.

nodes C and D. After these two scaling out operations, node A is responsible for all sub-regions in *DJ* except for *DJK* and *DJM*. Similarly, the sketch for the region *DJKJ* is moved out of node C into node E as a result of subsequent scale out operation. It should be noted that the depth and span of the distributed sketch are dynamic and are constantly evolving according to the workload and operating conditions.

As the tree grows, having parent nodes 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 nodes. This is depicted in Figure 4, where the stream ingester directly sends data to node E instead of sending it through nodes A and C. We use our gossiping subsystem to update parent nodes on child state information, which is useful for scaling in as explained in Section 3.4.

3.3 Coping with High Data Rates: Scaling out

There are two primary approaches to scaling a node that is experiencing high traffic: *replication* and *load migration*. In replication-based scaling, new nodes are spawned during high data arrival rates that are responsible for identical spatial scopes as their originating node. Assimilation of the newly-created node involves partitioning inbound streams directed to the original node. The upstream node is responsible for this partitioning, which may be performed in a skewed fashion with the new node receiving a larger portion of the inbound stream. Alternatively, inbound streams to the original node may also be partitioned in a round-robin fashion between the original and the newly-created node.

In targeted load migration, particular geospatial scopes are evicted from the original node to the newly created node during heavy traffic. Deciding which spatial scopes to migrate is based on data arrival rates and the rates at which particular portions of the sketch are being updated.

In Synopsis, we use targeted load migration for scaling out. Our implementation closely follows the MAPE loop [21] which is comprised of four phases: monitor (M), analyze (A), planning (P) and execution (E). A monitoring task periodically probes every node to gather two performance metrics:

1. Length of the backlog: This represents the number of

unprocessed messages in the queue. If the SYNOPSIS task cannot keep up with the incoming data rate, then the backlog will grow over time.

2. Memory pressure: Each node is allocated a fixed amount of memory. Exceeding these memory limits creates memory pressure, which may cause extended garbage collection cycles and increased paging activity. This will eventually lead to reduced performance in every SYNOPSIS task running on the node. The monitoring task records memory utilization of the process as well as its individual tasks.

The objective of scaling out is to maintain the *stability* of each node. We define stability as the ability to keep up with incoming data rates while incurring a manageable memory pressure. During the analysis phase, we use threshold-based rules [18] to provide scale out recommendations to SYNOPSIS nodes. We rely on a reactive scheme where the rules are evaluated on current observations. Scaling out recommendations are provided if either of the following rules are consistently satisfied for a certain number of observations:

- Backlog growth, which indicates that a portion of the load needs to be migrated to a different SYNOPSIS 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 SYNOPSIS node executes the planning and execution phases. During the planning phase, the node chooses portion(s) of the region within its current purview to be handed over to another node. For this task, the node relies on metadata it maintains for each subregion (corresponding to longer Geohash strings) and a numeric value provided by the scale out recommendation that measures how much load should be migrated. This metadata includes the data rate and the timestamp of the last processed message for each subregion. A SYNOPSIS node updates these metadata records with each message it processes. Nodes often migrate several prefixes during a scale out operation.

Only a single scale out operation takes place at a given time per node, 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 4 when node A is scaling out to node D . Once the SYNOPSIS node decides on subregions to scale, it initiates the scale out protocol by contacting the *deployer* node, which is responsible for launching tasks. In this message, it includes a list of preferred SYNOPSIS nodes for the load migration as well as memory requirements and expected message rate for the load. The preferred node set includes the SYNOPSIS nodes that already hold other subregions. The objective here is to minimize the number of nodes 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 node. Based on this view and the information present in the request, the deployer replies back with a set of target SYNOPSIS nodes. If a suitable node cannot be found, a new node will be launched that includes the location in question. Upon receiving a response from the deployer, the node that is scaling out contacts the target node and tries to acquire the mutex. A lock will be granted if the target can accommodate the load and no other scaling operations are taking place. If the lock acquisition fails, another node from the list is attempted; otherwise, the original SYNOPSIS node will create a pass-through channel and direct traffic towards the target node. Once this process is complete, the source node will initiate a state transfer asynchronously using a background channel to ensure the stream data flow is not affected, and update its memory utilization metrics to account for the pending state transfer.

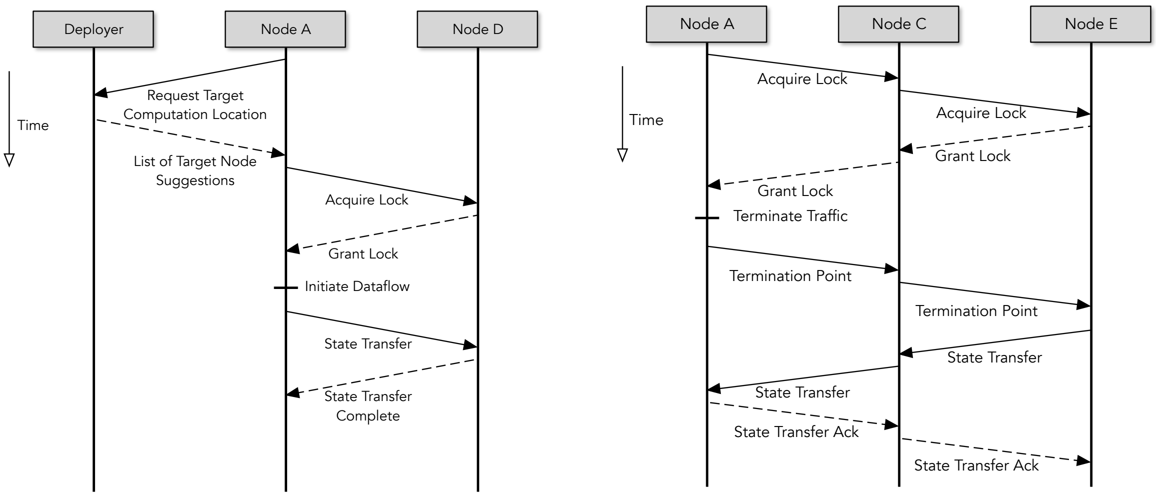
3.4 Scaling In

During scaling in, SYNOPSIS nodes 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 nodes. 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 are followed by a stabilization period.

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 SYNOPSIS node 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 node A in Figure 4 decides to scale in the subregion *DJK*, then it must communicate with both nodes C and E.

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 4 where node C is scaled in. As per our example, node A will have to acquire locks for nodes 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 nodes corresponding to this subregion is immediately terminated.

The state acquisition phase begins next. To ensure that SYNOPSIS does not lose any messages, the initiating node sends a *termination point* control message to the child node. The termination point is the sequence number of the last message sent to the child node either by the parent itself or by the short circuit channel. Once the child node has processed every message up to the termination point, it sends out termination point messages to all relevant child nodes. In our example, node C sends a termination point control message to node E upon processing the stream packet corre-



(a) Scale out protocol

(b) Scale in protocol

Figure 5: Steps of dynamic scaling protocols in chronological order

sponding to the termination point sent by node A. After the entire subtree has seen all messages up to the termination point, they acknowledge the initiator node and start transferring their states asynchronously. Once the parent node 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 releasing its lock after each child has released its lock.

3.5 Query Evaluations

Synopsis incorporates support for user-defined queries that are evaluated over the distributed sketch. Queries are described by the user in a SQL-like format or with JSON-based key-value descriptions similar to GraphQL [6]. Exact-match, range-based, and summarization queries are all supported over spatiotemporal bounds and individual feature values. Depending on scaling operations and the spatial scope of the queries, evaluations are carried out on one or more sketchlet instances. Information on the placement of sketchlets in the system and their corresponding feature scopes is maintained at each node 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 nodes comprising the distributed sketch. During query evaluations, the first step is to identify the set of Synopsis nodes that are relevant to the query. The conduit consults its prefix tree to locate nodes based on spatial, chronological, and feature constraints specified by the user. After this process is complete, the conduit forwards the queries on to the nodes for evaluation and supplies the client with a list of nodes that will respond to the query. As queries execute, their results are streamed back to the client where they are merged by the client API. This strategy ensures I/O and processing activities are interleaved on the client side.

Our distributed prefix tree enables query evaluations during both scaling in and out. When a conduit attempts to forward a query to a node that is undergoing a scaling operation, the request will be redirected to the destination's parent node. This process can continue recursively up through the network, ensuring queries will reach their final destination.

3.6 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. We classify queries – discrete or continuous – supported by Synopsis into 5 broad categories, including filter, statistical, density-based, set, and inferential.

3.6.1 Relational Queries

Relational queries describe the feature space in the context of our hierarchical graphs 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. Subsketches may be manipulated and inspected on the client side, and then reused in subsequent queries to request more detail or broaden the query scope. Relational queries can optionally return statistical metadata stored in the leaf nodes of the graph, which is also supported by our statistical query functionality.

3.6.2 Statistical Queries

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. Statistical queries also support retrieval of the mean, standard deviation, and feature outliers based on Chebyshev's inequality.

3.6.3 *Density Queries*

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. Kernel density estimations can request the function itself, an integral over a range of values, or the probability of a single value.

3.6.4 *Set Queries*

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).

To determine set membership, we use space-efficient bloom filters [2]. Bloom filters may produce false positives, but never false negatives. Besides returning a `true` or `false` result to the user, membership queries also include the probability of the answer being a false positive. Set cardinality (number of distinct elements) queries are supported by the HyperLogLog [7] algorithm. HyperLogLog is able to estimate cardinality with high accuracy and low memory consumption. Finally, observation frequencies are provided by the count-min data structure [5]. Count-min is structurally similar to a bloom filter, but can be used to estimate the frequency of values within a particular error band. Frequency queries are accompanied by their associated confidence intervals and relative error.

3.6.5 *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 graph can produce two-dimensional regression models that forecast new outcomes across each feature type when an independent variable of interest changes. In their continuous form, inferential queries are backed by machine learning models that are *installed* for a particular time window and can be trained using either the sketch or new, full-resolution values as they arrive. Continuous inferential queries can stream predictions back to the client on a regular interval, or a subsequent query can reference a particular model and parameterize it to make a single prediction. Our current implementation of SYNOPSIS supports multiple linear regression, but our machine learning interface allows new models to be plugged into the system at run time.

3.7 Coping with Failures in Synopsis

SYNOPSIS relies on passive replication to recover from node 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 SYNOPSIS node depends on the entire set of stream packets it has processed previously [4].

Support for fault tolerance is implemented by augmenting the data processing graph (which is responsible for updating the distributed sketch) with a set of state replication graphs each attached to a SYNOPSIS node. The state replication graph is a two-stage graph where the first stage (*state replication source*) periodically sends changes to the state of its associated SYNOPSIS node since its last message to the stage two operator. This incremental checkpointing scheme forms an edit stream between the two stages. This consumes a less bandwidth compared to a periodic checkpointing scheme that replicates the entire state every time [4]. The stage two operator, which is also known as the *state replication sink*, serializes the messages received via the edit stream to persistent storage. By default, SYNOPSIS uses the disk of the machine which hosts the state replication sink operator for persistent storage, but necessary API level provisions are included to support highly available storage implementations such as HDFS [3]. The state replication sink operator can fan out to more instances to support higher replication levels. For instance, if the required replication level is three then there will be three instances of the state replication sink running. Having a fixed number of state replication sinks and not using any additional memory to maintain state replicas contribute to a smaller resource footprint required for fault tolerance.

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.

SYNOPSIS nodes are implemented as idempotent operators (using the message sequence numbers), hence they will process the replayed messages only if necessary. Users can apply their own policy for defining the period between the incremental checkpoints because too frequent checkpoints can incur high overhead whereas longer periods between successive checkpoints may consume more resources for upstream backups as well as for processing replayed messages in case of a failure.

Membership management is implemented using Zookeeper [15], which is leveraged to detect failed nodes. Upon receiving notifications from the system about node failures, an upstream node switches to a secondary child node if necessary. The secondary child node is a SYNOPSIS node running on the process where a state replication sink was running; allowing it access to the persisted replicated state. 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 messages and support for merging with other sketches is useful during this failure recovery process.

3.8 Visualization

To demonstrate the potential applications of Synopsis, we created two representative 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 6 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.

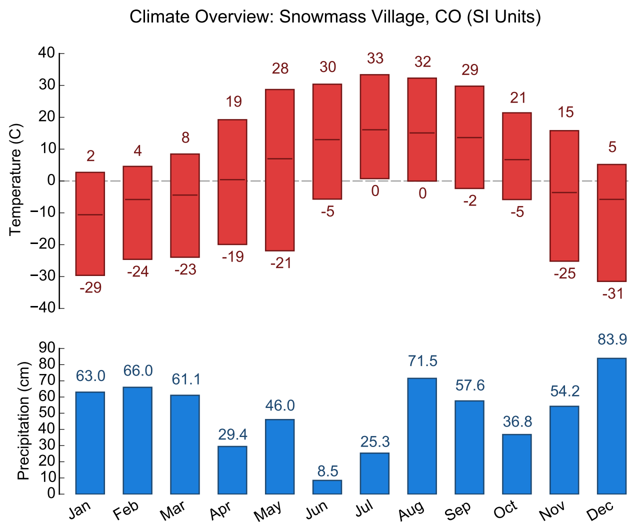


Figure 6: Climate chart visualization

Our second visualization ...

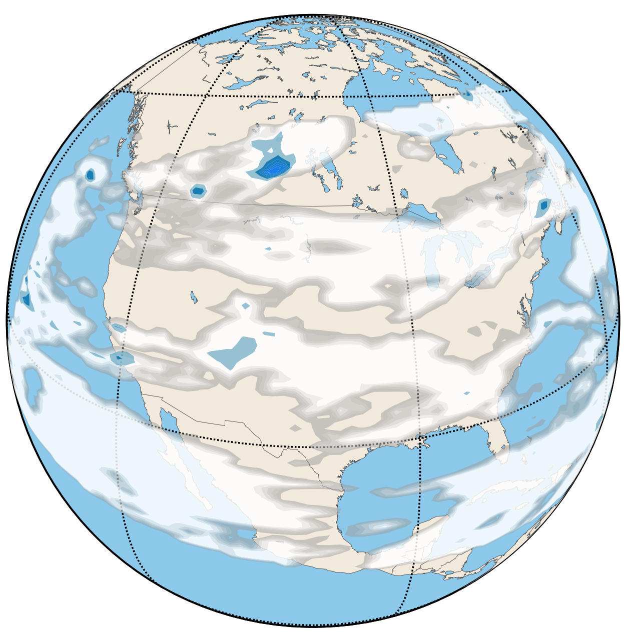


Figure 7: Global contour visualization

4. PERFORMANCE EVALUATION

To evaluate SYNOPSIS, we used a real-world dataset to populate the distributed sketch. This includes dynamic scaling and an analysis of node stability as scaling operations

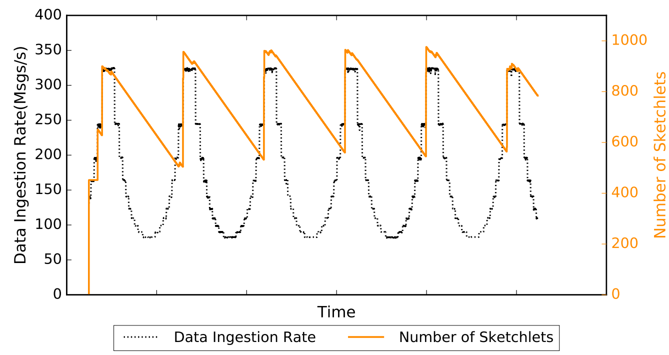


Figure 8: The variation of number of sketchlets with the data ingestion rate.

take place, as well as query evaluation latencies for each query type.

4.1 Dataset and Experimental Setup

Our subject dataset was sourced from the NOAA North American Mesoscale (NAM) Forecast System [23]. 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.

Our performance evaluation was carried out on a 75-node heterogeneous cluster consisting of 45 HP DL160 servers (Xeon E5620, 12 GB RAM) and 30 HP DL320e servers (Xeon E3-1220 V2, 8 GB RAM) running Fedora 23. SYNOPSIS was executed under the OpenJDK Java runtime, version 1.8.0_72.

4.2 Dynamic Scaling

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 less than the highest possible throughput that will create a backlog at SYNOPSIS nodes. 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 in child nodes after the targeted load migration. We started with a single SYNOPSIS node 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 a rapid scale out activity during high data ingestion rates whereas scaling in takes place gradually with one sub region (hence one sketch) at a time.

Figure 9 visualizes a snapshot of the stream processing graph in runtime which validates our dynamic scaling implementation. 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 SYNOPSIS nodes 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

Table 2: Generating a random forest based regression model using Spark Mllib

Dataset	Size (GB)	Data Loading Time (s)		Model Training Time (s)		Accuracy (RMSE)	
		Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Original Data	25.350	4.611	0.133	168.803	9.673	6.191	0.005
Synthetic Data - 10%	2.549	3.424	0.288	125.527	7.145	6.210	0.013
Synthetic Data - 20%	-	-	-	144.681	8.2845	6.196	0.015

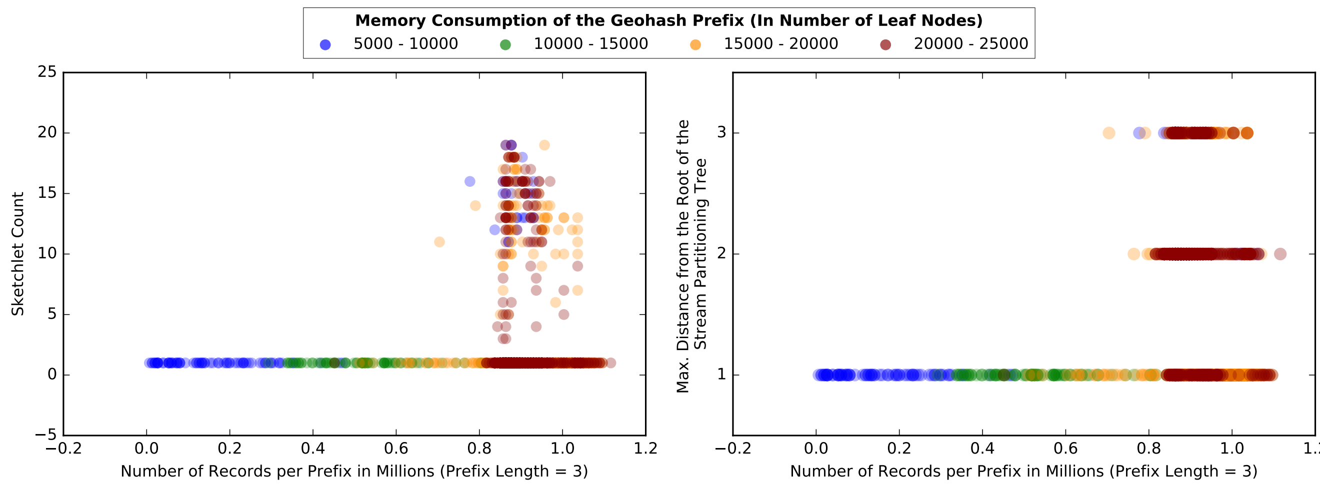


Figure 9: Analysis of a snapshot of the stream processing graph during data ingestion demonstrating the size and distribution of the information corresponding to different prefixes against the observed record count. If the information is dispersed over multiple sketchlets, it is likely to be a prefix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

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 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 nodes which results in a higher distance from the root in 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.

4.3 Stability at Individual Nodes

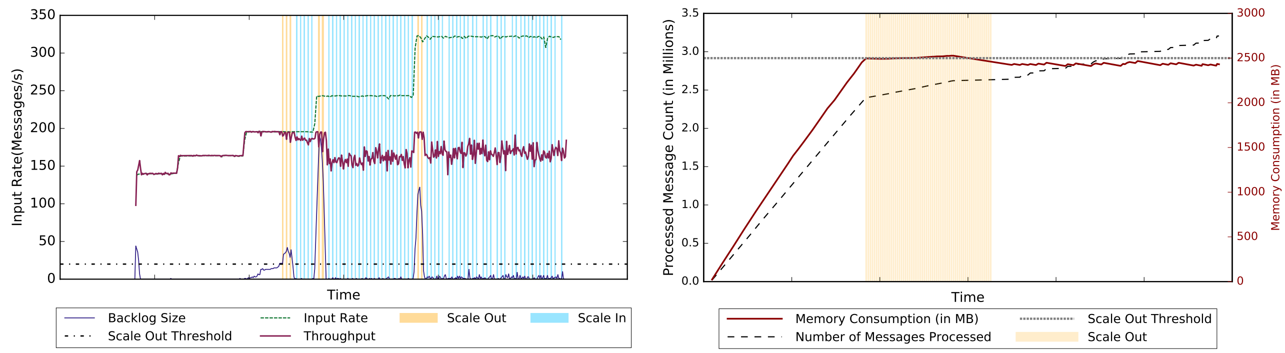
The objective of this benchmark was to demonstrate how scaling out operations manage to maintain stability at each node under varying workload conditions. The same setup as in previous micro benchmark was used, but the evaluation metrics captured are corresponding to an individual node instead of the entire system. 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 captured how backlog length and throughput at an individual node vary with the input rate when dynamic scaling is enabled. The SYNOPSIS node that was considered for the experiment immediately received data from stream ingesters, hence the input rate observed at the node closely resembled the varying data ingestion rate. As shown in Figure 10a, scaling out helps a node 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 scaling out and continuous, gradual scaling in as explained in section 3.3.

Figure 10b demonstrates how memory consumption based threshold-based rules trigger scaling maneuvers to maintain the system stability. 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.

4.4 Growth of the Distributed Sketch over Time

We evaluated the growth in memory consumption of the distributed sketch over time with continous data ingestion as shown in Figure 11. The rate of growth of the distributed sketch is decreased over time. At the end of our monitoring



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

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

Figure 10: Scaling out enables maintaining stability at an individual node based on backlog growth and memory usage

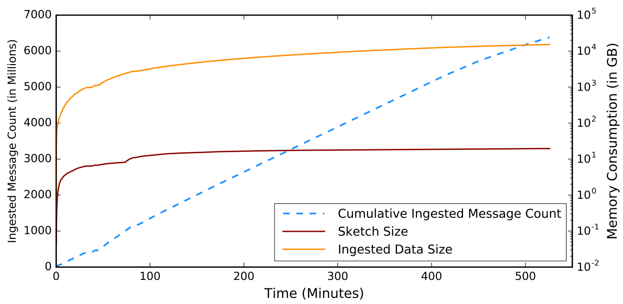


Figure 11: Memory usage of the distributed sketch over time against the amount of ingested data. The rate of growth of the distributed sketch decreases over time resulting in a sketch which is multiple degrees of magnitudes smaller than the amount of ingested data

period, the ingested data size was over three magnitudes higher (~ 1285) than the sketch size.

4.5 Sketch Query Evaluation

To evaluate the performance of the sketch, we executed several representative query workloads across a variety of sketchlet sizes. These queries were separated into two groups: data structure lookups and graph lookups. Data structure lookups include density queries, set queries, and summary statistics for the entire sketch, while graph lookups involve targeted portions of the overall feature space. In general, queries that require a graph lookup consume more processing time, but are much more expressive; for instance, such 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, while a data structure lookup would be restricted to chronological parameters and select features of interest. Table 3 outlines the query times for both evaluation types. In general, data structure lookup operations require minimal processing. While graph 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.

Table 3: Query 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
Statistics	0.002	0.001
Subgraph Stat. (5 km)	46.357	1.287
Relational (5 km)	40.510	6.937
Relational (25 km)	47.619	6.355
Relational (800 km)	53.620	6.818

Figure 12 demonstrates the efficiency of the query evaluations over the distributed sketch. Cumulative query throughput and latencies were measured with different number of concurrent query funnels. A query funnel continously generates and dispatches random queries at their maximum possible rate. Our setup included 40 Synopsis nodes and one of those nodes was randomly chosen as the conduit for each query.

5. RELATED WORK

Data Cubes [8, 11, 22, 14] 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

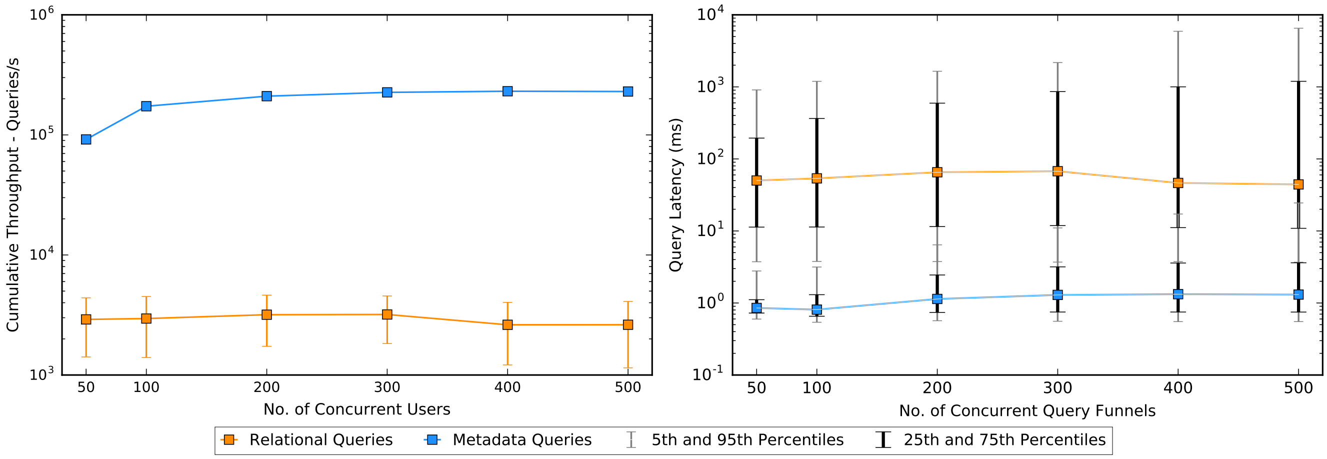


Figure 12: Distributed Query Evaluation Performance. Variation of cumulative throughput and latency against the number of concurrent query funnels in a 40 node Synopsis cluster.

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 [10] 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 [20, 19] is a distributed hash table that supports the storage and retrieval of multidimensional data. Given the 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) [28] extends Spark SQL [1] to support spatial operations as first class members in SQL as well as in DataFrames. It relies on data to be stored in Spark [29]. Even though this approach provides a high accuracy, it is not scalable for geospatial streams in long term due to demanding storage requirements. In Synopsis, spatial queries can be executed with a reasonable accuracy without having to store streaming data as it is.

Dynamic scaling and elasticity in stream processing systems has been studied thoroughly [13, 9, 4, 17, 12, 25]. Heinze et al. [13] explores using different dynamic scaling schemes including threshold-based rules and reinforcement learning using the FUGU [12] 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.

StreamCloud [9] relies on a global threshold-based scheme to implement elasticity where a query is partitioned into sub-queries 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 node starts building the state (sketch) which is later merged with the asynchronously transferred state from the previous node.

Gedik et al. [25] 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 stream partitioners 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 nodes with lower resource utilization.

Our state migration scheme doesn't require migrating states between nodes 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.

6. CONCLUSIONS AND FUTURE WORK

Our framework for constructing a distributed sketch over spatiotemporal streams, SYNOPSIS, maintains a compact representation of the observational space, allows dynamic scaling in and out to preserve responsiveness and avoid overprovisioning, and 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 graph 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 scaling in and out (**RQ-2**). During query evaluations, only the SYNOPSIS nodes 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 that are expressed as MapReduce jobs. 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 has been supported by funding (HSHQDC-13-C-B0018 and D15PC00279) from the US Department of Homeland Security, the US National Science Foundation's Computer Systems Research Program (CNS-1253908), and the Environmental Defense Fund (0164-000000-10410-100).

7. REFERENCES

[1] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, 2015.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] D. Borthakur. Hdfs architecture guide. *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, 2008.

[4] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch. 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*, pages 725–736. ACM, 2013.

[5] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, Apr. 2005.

[6] Facebook, Inc. GraphQL, 2015.

[7] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm. In *AOFA '07: Proceedings of the 2007 International Conference on the Analysis of Algorithms.*, 2007.

[8] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proceedings of the Twelfth International Conference on Data Engineering*, ICDE '96, pages 152–159, Washington, DC, USA, 1996. IEEE Computer Society.

[9] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez. Streamcloud: An elastic and scalable data streaming system. *Parallel and Distributed Systems, IEEE Transactions on*, 23(12):2351–2365, 2012.

[10] C. Gupta, S. Wang, I. Ari, M. Hao, U. Dayal, A. Mehta, M. Marwah, and R. Sharma. Chaos: A data stream analysis architecture for enterprise applications. In *2009 IEEE Conference on Commerce and Enterprise Computing*, pages 33–40, July 2009.

[11] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 205–216, New York, NY, USA, 1996. ACM.

[12] T. Heinze, Y. Ji, Y. Pan, F. J. Grueneberger, Z. Jerzak, and C. Fetzer. Elastic complex event processing under varying query load. In *BD3@ VLDB*, pages 25–30. Citeseer, 2013.

[13] T. Heinze, V. Pappalardo, Z. Jerzak, and C. Fetzer. Auto-scaling techniques for elastic data stream processing. In *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on*, pages 296–302. IEEE, 2014.

[14] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in olap data cubes. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 73–88, New York, NY, USA, 1997. ACM.

[15] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference*, volume 8, page 9, 2010.

[16] M. Kristan, A. Leonardis, and D. Skocaj. Multivariate online kernel density estimation with gaussian kernels. *Pattern Recognition*, 44(10âĂŞ11):2630 – 2642, 2011. Semi-Supervised Learning for Visual Content Analysis and Understanding.

[17] S. Loesing, M. Hentschel, T. Kraska, and D. Kossmann. Stormy: an elastic and highly available streaming service in the cloud. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 55–60. ACM, 2012.

[18] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano. 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*, 12:2012, 2012.

[19] M. Malensek, S. Pallickara, and S. Pallickara. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2015.

[20] 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*, PP(99):1–1, 2016.

[21] M. Maurer, I. Breskovic, V. C. Emeakaroha, and I. Brandic. Revealing the MAPE loop for the autonomic management of cloud infrastructures. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 147–152. IEEE, 2011.

[22] I. S. Mumick, D. Quass, and B. S. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 100–111, New York, NY, USA, 1997. ACM.

[23] National Oceanic and Atmospheric Administration. The North American Mesoscale Forecast System, 2016.

[24] G. Niemeyer. Geohash, 2008.

[25] S. Schneider, H. Andrade, B. Gedik, A. Biem, and K.-L. Wu. Elastic scaling of data parallel operators in stream processing. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12. IEEE, 2009.

[26] B. W. Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

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

[28] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo. Simba: Efficient in-memory spatial analytics.

[29] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.