Rivulet: A Distributed Sketch for Voluminous Spatio-Temporal Observational Streams to Support High Throughput Query Evaluations

Name1, Name2, Name3, Name4
Department of Computer Science
Colorado State University
Fort Collins, Colorado, USA
{name1,name2,name3,name4}@cs.colostate.edu

ABSTRACT

Abstract goes here.

1. INTRODUCTION

The proliferation of remote sensing equipment (such as radars and satellites); networked sensors and monitoring equipment; and commercial applications such as mapping services, location-based recommendation services and sales tracking have resulted in the exponential growth of spatiotemporal data. Spatiotemporal data comprise observations where in addition to the features of interest (such as humidity, air quality, disease prevalence, sales, etc) both the spatial and chronological components representing the time and location where measurements were made are available.

Spatiotemporal data includes a wealth of information that can be used to inform: decision making, scientific modeling and simulations, and resource allocations in several domains. These domains include atmospheric science, epidemiology, environmental science, geosciences, smart-city settings, and commercial applications. Querying these voluminous datasets and the expressiveness of these queries underpin efficiency in these settings. To be effective, regardless of the data volumes, the query evaluations must be in real-time and at high-throughput meaning a large number of queries must be evaluated per second.

Spatiotemporal datasets are naturally multidimensional with multiple features of interest being reported/recorded continually for a particular timestamp and geolocation. The values associated with these features are continually changing — in other words, the feature space represented by these datasets are continually evolving.

Queries specified over these datasets may have wide range of characteristics encompassing the frequency at which they are evaluated, and the spatiotemporal scope associated with these queries. The crux of this paper is to support query evaluations over continually arriving observational data. The queries may be continuous or discrete, involve sliding windows, and 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.
- Accuracy: Queries specified by the user continuous or one-time must be accurate.
- Continuous query evaluations: Users should be able to register query evaluations over the observational space that must be continually evaluated.
- 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

1. How can we generate compact, memory-resident (i.e. the sketch) representations of the observational space? The sketch must be amenable to fast, continuous updates to ensure that the sketch is representative of the observational space.
2. How can we scale effectively in situations where the observations arrive at a rate faster than the rate at which the sketch can be updated? The density and arrival rates for observations vary based on the geospatial characteristics of the data. For example, in a smart-city setting NYC would like have a far higher rate of observations than a mid-sized city like Denver, which in turn would have a far higher rate of observations than a small city such as Fort Collins.
3. How can we effectively account for the spatiotemporal attributes associated with both the observations and

the queries specified by the users? This spans the generation and updates of the sketch, scaling in response to high data arrival rates, and the evaluation of the queries.

4. How can we ensure high-throughputs in the assimilation of observations, updates to the sketch, and query evaluations? A particular challenge is preservation of accuracy in query evaluations despite high data arrivals, scaling in response to changes in system load, and concurrent query evaluations. This encompasses using horizontal scaling to proactively alleviate hotspots, maintaining a distributed representation of the sketch, and effective distribution, targeting, and combining of query evaluations.

1.3 Methodology

Our methodology targets: (1) creation of the sketch, (2) updating the sketch in response to data arrivals, (3) identifying performance hotspots via targeted alleviation of performance hotspots, (4) support for splitting and fusing a sketch, and (5) dynamic creation of distributed sketch. (6) using the distributed sketch to perform query evaluations, and (7) achieve high-throughput by minimizing synchronization between different portions of the distributed sketch.

Extract metadata from observations Organize this metadata in an in-memory sketch.

Our sketch supports 3 key features. Specifically, the sketch (1) maintains a compact representation of the observations seen so far, (2) is amenable to splitting and fusing, and (3) supports query evaluations including specification of sliding window sizes and geospatial scopes to support exploring properties of the observational space. The sketch is naturally amenable to support distributed representations; with each node within the cluster holding information about the observational space. The distributed representation of the sketch is that of a dynamic, continually-updated in-memory store.

There are several advantages to a distributed representation of the sketch. (1) It allows us to maintain a richer (finer-grained) representation of the feature space. (2) it improves accuracy during query evaluations, (3) supports multiple, concurrent query evaluations with each node evaluating queries independently.

Furthermore, at each node we dynamically adjust the representativeness of the sketch. There are two options here: bias towards the recent past or equal representations. 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. For example, last 3 minutes, last 3 hours, last 3 days, last 3 months, last 3 years, and last 3 decades would have equal memory footprints. In the latter approach, the sketch maintains a coarser-grained representations at each node.

The sketch scales (up or down) dynamically to cope with data arrival rates. At each node, the expressiveness of the sketch can be tuned dynamically based on the available memory. At each node, the sketch continually updates the expressiveness of the portions of the feature space, dynamically reorients the graph-based structure, and minimizes traversals (using Bloom Filters) to support faster evaluations and preserve high-throughput.

1.4 Paper Contributions

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

- Design of the Trident sketch that maintains compact, distributed representations of the observational space. At each node, the sketch supports dynamic reorientations to conserve memory and support faster query evaluations.
- Support for dynamic scaling of the sketch in response to the data arrival rates. Upscaling operations support targeted alleviation of hotspots. Only geographic locations that are observing high data arrival rates are scale. The framework manages the complexity of identifying these hotspots, splitting the particular portion of the sketch, and migrating relevant portions of the sketch. Most importantly, the framework achieves this while maintaining acceptable levels of accuracy in query evaluations.
- Support for one-time and continuous evaluations of the observational space. We incorporate support for CQL queries. Query evaluations can target arbitrarily sized (chronological) window sizes and geospatial scopes.
- Support for high-throughput, distributed evaluation of queries. To our knowledge, Trident 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...

2. SYSTEM OVERVIEW

Rivulet is implemented on top of Neptune stream processing system [1] as a specialized layer for geo-spatial data. In this section, we will briefly introduce Neptune followed by a discussion on the design of Rivulet.

2.1 Neptune

Neptune is a distributed stream processing system which is optimized to process high throughput data streams. It is designed to cope with application requirements such as high volumes of small stream packets, high and variable data rates and heterogeneity within stream processing jobs. Satisfying these requirements imposes various system level challenges including possible buffer-overflow errors, increased number of context-switches, object creation overhead and memory management issues. Neptune takes a holistic approach that considers CPU, memory, network and kernel issues to address the above challenges. This is achieved through employing a set of optimizations such as application level buffering, batched scheduling, object re-use, built-in support for backpressure and selective compression to ensure a better utilization of system resources.

Users can deploy stream processing jobs as directed acyclic graphs in Neptune. A stream processing graph consists of

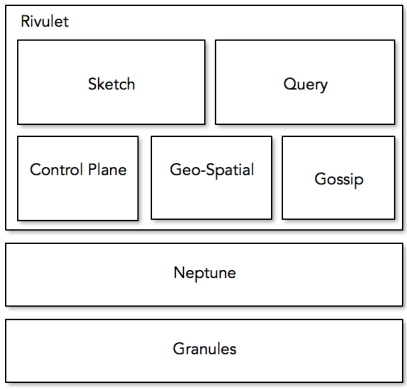

Figure 1: Rivulet is implemented as a specialized layer for geo-spatial data on top of Neptune stream processing system.

a set of stream ingestion points and stream processors (vertice) and streams connecting these operators (edges). To provide each job with a initial load-balanced deployment plan to start with, each operator in a stream processing graph can be annotated with a parallelism level while each stream can be associated with a partitioning scheme.

2.2 Rivulet

Rivulet is a stream processing graph deployed on top of Neptune. It is comprised of several stream ingestion operators and a dynamic set of stateful stream processors. These stream processors collectively maintain the distributed sketch while the number of processors are dynamically adjusted by the Rivulet runtime to suit the changing operating conditions. We use a stream partitioning scheme based on Geohash of stream packets to route stream packets to the appropriate stream processor which holds the corresponding section of the distributed sketch.

Rivulet extends the Neptune's generic stream processor to provide a set of auxiliary services required by its runtime. These services are control plane, gossip subsystem and querying subsystem as depicted in Figure 1.

Control plane: Control plane is responsible for orchestrating control messages exchanged between Rivulet nodes as part of various distributed protocols such dynamic scaling. Control plane is decoupled from the data plane to ensure high priority and low latent processing without being affected by buffering delays and backpressure.

Gossip subsystem: Majority of the Rivulet functionality is assumed to work with the local knowledge of a particular node, certain functionalities require an approximate global knowledge. For instance, each Rivulet node maintains a prefix tree, to help the querying subsystem to know which nodes are holding the sections of the sketch responsible for the geohash locations corresponding to a query it evaluates (This is further explained in section 4.5). In order to establish and maintain this global view of the entire system, Rivulet nodes gossip about their state periodically as well as when there is a change in the state which other nodes in the system are interested in. Given that there is a

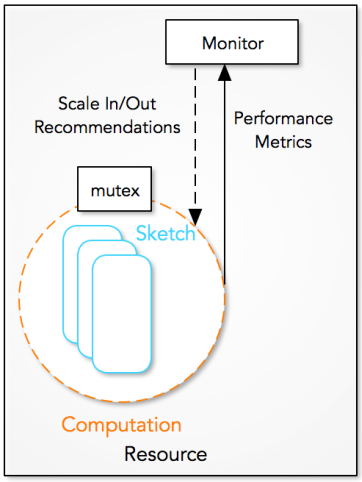

Figure 2: Dynamic scaling is triggered by monitoring the backlog of each computation and memory pressure incurred in each process.

propagation and convergence delay inherent with the gossip protocols, at any given time, a Rivulet node will only have an eventually consistent view of the entire system.

Querying subsystem: Querying subsystem is responsible for...

Each Neptune process in the system host one or more Rivulet nodes. A Rivulet node is essentially a task triggered to on the availability of data in any of its incoming streams. Each Rivulet node is responsible for one of more sections of the distributed sketch as illustrated in Figure 2. All Rivulet nodes running in a particular Neptune process is constantly probed by a monitoring task to gather their performance metrics: backlog information and memory utilization by individual sketches. This information is used for dynamic scaling recommendations as explained in section 4.3.

3. SYNOPSIS CONSTRUCTION

Our *Synopsis* data structure is a multidimensional, graph-based model of incoming data streams. Each resource in the system maintains an in-memory Synopsis 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 Graph Structure

Synopsis 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

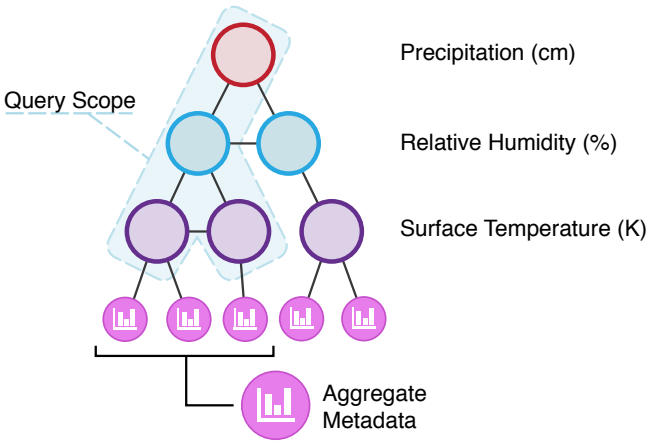

Figure 3: A simplified Synopsis instance with a three-plane hierarchy and sample query scope, leading to several metadata leaves. In production settings, a Synopsis will contain hundreds of thousands of vertices and edges.

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 3 demonstrates the structure of a Synopsis 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 Synopsis.

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 [8]. 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 Synopses on disparate resources and then merged into a single, coherent result Synopsis.

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 a low variety of possible values 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 Synopsis 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.

3.2 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 Synopsis 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.

To determine the size and quantity of these bins, Synopsis instances maintain additional graph metadata provided by the multivariate online kernel density estimation (oKDE) algorithm developed by Kristan et al. [2]. 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 [7]. 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 test dataset: 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 approximately 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: Graph statistics (before/after quant)

Configuration	Vertices	Edges	Leaves
A			
B			
C			

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 features from the Synopsis 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.

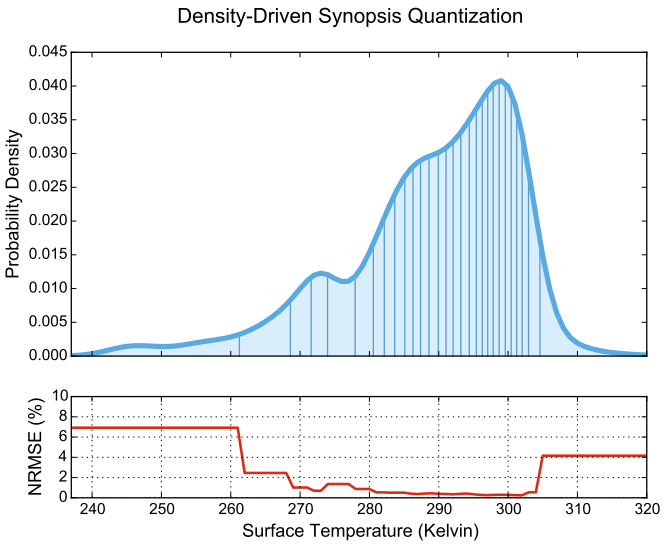

Figure 4: A demonstration of the quantization process, with 29 vertex bins generated across the distribution of surface temperature values in our dataset. Each bin is indicated by a vertical line under the curve.

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 feature planes are removed from the graph 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.

Table 2: Graph statistics (after temp. reduction)

Synopsis Age	Vertices	Edges	Leaves
1 month			
6 months			
12 months			

4. METHODOLOGY

4.1 Construction of the Sketch

Metadata extraction and organization
Fine-grained/coarse-grained representations
Dynamic reorientations of the graph
Representativeness across time intervals

4.2 Stream Partitioning

We use the Geohash algorithm [6] to balance load and partition incoming data streams across processing resources.

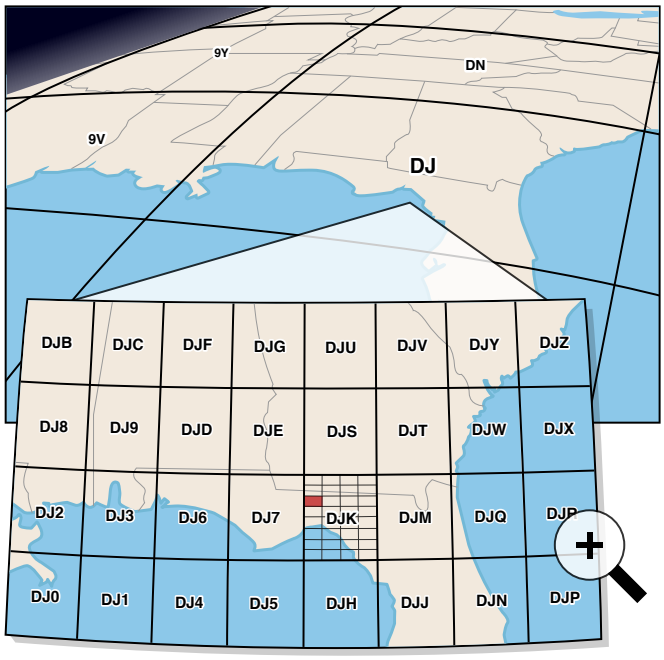

Figure 5: A demonstration of the Geohash algorithm. Each additional character in a Geohash string describes a finer-grained region; Geohash *DJ* contains substantial parts of Florida, Georgia, and Alabama, USA, while *DJKJ* (highlighted in red) encompasses Tallahassee, the capital of Florida.

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 5 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 Rivulet 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. This property allows us to manage and allocate resources across a wide variety of observational densities.

Figure 6 depicts a possible arrangement of the distributed sketch and the associated stream partitioning scheme corresponding to the same geographic region described above. The distributed sketch is arranged in a tree-like structure. Stream ingesters act as root nodes of this tree structure and they partition the geo-spatial stream among the Rivulet nodes using the Geohash based partitioning function. Nodes closer to the root hold sketches corresponding to shorter Geohash strings. In other words, they contain sketches for larger geographical regions. For instance, nodes A and B

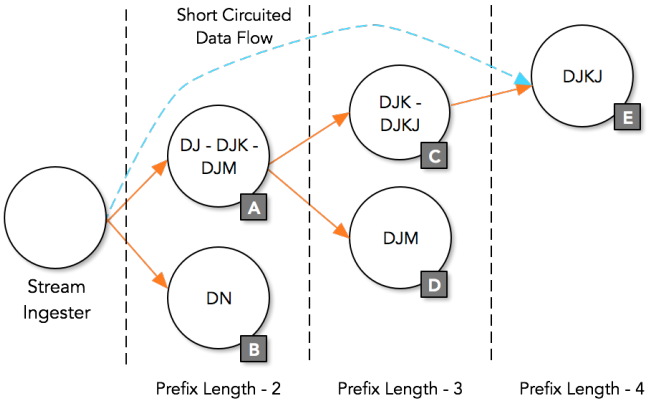

Figure 6: An example of the stream partitioning based on Geohashes of the stream packets

in Figure 6 are responsible for regions represented by Geohashes *DJ* and and *DN* respectively. Node E, which is 3 edges deep from the stream ingester, is responsible for a smaller region, *DJKJ*.

Rivulet is designed to ensure that the regions which are corresponding to larger portions of input streams, hence more frequent updates to the sketch, 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 (sketch and related meta data) 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 4.3). More specifically, it will not process the stream packet, instead updates its statistics based on the headers of the packet and forwards it to the child node . For instance, node A in Figure 6 have scaled out two regions (*DJK* and *DJM*) to 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.

When the height of the tree grows, it becomes inefficient to have parent nodes passing through the traffic intended for the child nodes. Because it consumes extra bandwidth as well as increases the latency of the corresponding sketch updates due to increased number of hops the packets have to travel through. To circumvent this, we use short circuiting to direct traffic from stream ingesters straight to child nodes and avoid routing through parents. This is depicted in Figure 6 where stream ingester directly sends data to node E instead of sending it through nodes A and C. We use our gossiping subsystem to update parents on the statistics related to child prefixes which are useful for scaling in decisions at later points of time as explained in section 4.3. Due to short circuiting, the data flow path starts to deviate from the tree like structure of the distributed sketch.

4.3 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, during high data arrival rates, the original node spawns a new node that is responsible for identical spatial scopes as the original. 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, during heavy traffic the original node spawns a new node. However, in this case, particular geospatial scopes are evicted from the original node to the newly created node. The decision about the spatial scopes to be migrated is based on the data arrival rates and the rates at which particular portions of the sketch are being updated.

Replication-based scaling introduces a challenge during query evaluations in that the query must be forwarded to all nodes responsible for a particular scope and the results merged; depending on the nature of these queries (for e.g., correlation analysis and inferential queries) merging of results may be difficult to accomplish without extensive state synchronizations.

In Rivulet, we use targeted load migration for scaling out. Our implementation closely follows the MAPE loop [4] which is comprised of four phases: monitor (M), analyze (A), planning (P) and execution (E). The monitoring task as shown in Figure 2 periodically probes every Rivulet task to gather two performance metrics as part of monitoring phase.

1. Length of the backlog: This represents the number of unprocessed messages in the queue. If the Rivulet task cannot keep up with the incoming data rate, then the backlog will grow over time.

2. Memory pressure: Each Neptune resource is a JVM process which is being allocated a fixed amount of memory. Exceeding these memory limits create memory pressure which may cause extended garbage collection cycles and increased paging activity. This will eventually lead to reduced performance in every Rivulet task running in the Neptune resource. Monitoring task will continuously record the memory utilization by the entire JVM process as well as the memory utilization by individual Rivulet tasks.

During the analysis phase, we use threshold based rules [3] to provide scale out recommendations to Rivulet nodes if necessary. Currently we rely on a reactive scheme where we evaluate the threshold based rules based on the current observations. Scaling out recommendations are provided if either of following rules are consistently satisfied for certain number of observations.

- Growing backlog - This is an indication that a portion of the load needs to be migrated to a different Rivulet node.
- High overall memory utilization above a certain threshold (threshold is usually set below the memory limits allowing a capacity buffer for the process to avoid oscillation)

Upon receiving a scaling out recommendation from the monitoring task, a Rivulet node executes planning and execution phases. During the planning phase, it will choose

the portion(s) of the region within its current purview to be handed over to another node. For this task, it relies on meta-data it maintains for each sub region (region corresponding to a longer Geohash string) and a numeric value provided along with the scale out recommendation which is a measure of how much load should be migrated. This meta data includes the data rate and the timestamp of the last processed message for each sub region. A Rivulet node updates these meta data with each message it processes. Often a Rivulet node migrates several prefixes during a single scale out operation.

Only a single scale out operation takes place at a given time in a Neptune process. This is controlled using a mutual exclusive lock (mutex) located in each Neptune process. 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 above constraints is to avoid oscillations in scaling activities. For instance, if system aggressively scales out in the presence of a memory pressure without allowing a stabilization period, it is possible that the system ends up in a state where it is under provisioned. As a result of this, it will start aggressively scaling in and run again into an over provisioned state.

Figure 7 depicts the phases of the scale out protocol. Once a Rivulet node decides on the sub regions, it initiates the scale out protocol by contacting the deployer. In this message, it includes a list of preferred Rivulet nodes for the load migration as well as memory requirements and expected message rate for the load. The preferred node set includes the Rivulet nodes that already holds its sub regions. The objective is to minimize the number of Rivulet nodes responsible for holding sketches for a given geographical region, because it reduces the number of Rivulet nodes to contact during a query evaluation for that region. Deployer has an approximate view of the entire system gathered through gossip messages which includes the memory pressure and cumulative backlog information of each node. Based on this view and the information present in the request, deployer replies back with a set of target Rivulet nodes. If it cannot find a suitable node out of the existing set, the deployer will launch a new Rivulet node and include its location in the new request. Upon receiving the response from the deployer, Rivulet node contacts the target node and try to acquire the mutex. Lock will be granted if no other scaling operations takes place in the Neptune process which holds the Rivulet node and it can accommodate the migrated load. The second condition is checked again because the deployer may not have most upto-date metrics regarding the target Rivulet node due to the eventual consistent nature of its system view. If the lock acquisition is failed, another node from the list of attempted. Else the original Rivulet node will create a pass-through channel and direct traffic towards the target node. In the same time, it will initiate a state transfer using a different channel in the background. This will ensure that the state transfer doesn't affect the stream data flow and it happens asynchronously and the protocol ends without waiting for state transfer to complete. Even if the state transfer is not complete, the target Rivulet node updates its memory utilization metric to account for the pending state transfer. Ending protocol within a short period of time is important because it can release mutual exclusive locks in both origin and target Rivulet nodes quickly and participate in other scaling activities soon after the stabilization period.

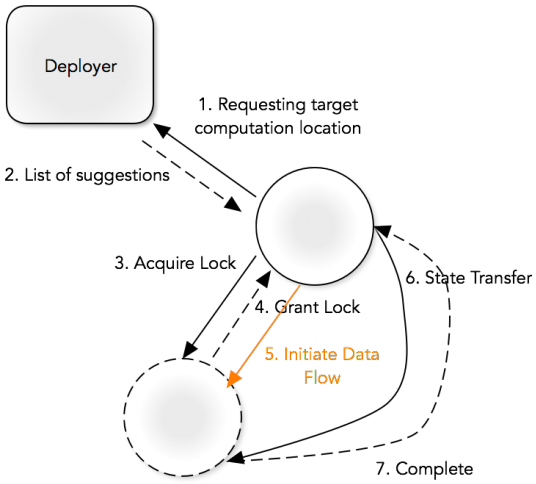

Figure 7: Scale out protocol

4.4 Downscaling

During Scaling in, Rivulet nodes attempt to take back some of the sub-regions it scaled out previously to other Rivulet nodes. This ensures better resource utilization in the system in addition to efficient query evaluations by having to contact less number of nodes. Scaling in operations are also guarded by the same mutual exclusive lock used for scale out (only one scale out or scale in operation takes place at a given time) and followed by a stabilization period.

Monitoring and analysis phases are similar to scaling out scenario 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 sub region is acquired during a single scale in operation. Scaling in is more complex than scaling out because it deals with more than one Rivulet node in most cases. Because at this point, it is possible that further scale out operations have taken place in the scaled out sub region after the initial scale out. For instance, if node A in Figure 6 decides to scale in the sub region *DJK*, then it has to work with both nodes C and E. Scale in protocol starts with a lock acquisition protocol similar to scale out protocol as illustrated in Figure 8. But it is required to lock the entire subtree where the sketch for the given sub region is distributed across. 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 sub tree, then the scale in operation is aborted and the monitoring process will start the next iteration of the MAPE loop immediately. If the sub tree is successfully locked, then data flow to the child nodes corresponding to this sub region is immediately terminated. If there was a short circuit set up before, it will be removed and the data will start flow to the parent node. The state acquisition phase begins next. To ensure that Rivulet does not lose any messages, the initiator node sends a termina-

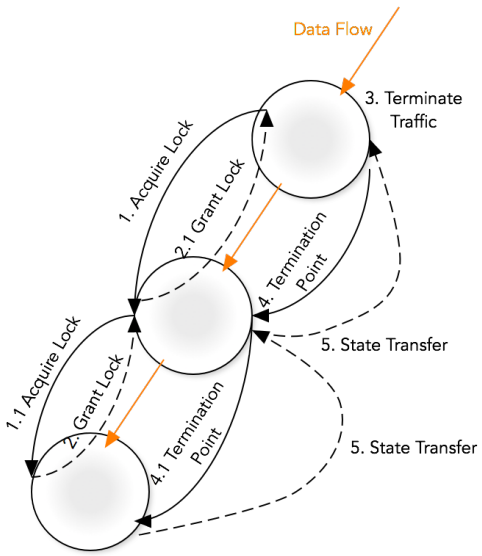

Figure 8: Scale in protocol

tion point control message to the child node. 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. It may be possible that the child has already processed this message and updated its sketch by the time it receives the termination point control message. But in extreme cases, the termination point control message may get processed before the actual stream packet with the same sequence number. This is because control plane and data plane use separate channels and also due to the possibility of data plane messages are being queued before processing. 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 corresponding to the termination point sent by node A). After the entire sub tree has seen all messages up to the termination point, they acknowledge the initiator node and starts transferring their states asynchronously as similar to scale out protocol. 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 sub tree where parent releases its lock after noticing that every child participated in the scale in protocol has released its lock.

4.5 Query Evaluations

Accounting for data arrival rates, accuracy requirements, memory pressures, and the capacity of network links result in dynamic upscaling and downscaling of nodes that comprise Rivulet. The result of these scaling operations is a distributed sketch, with constituent nodes holding compact representations of observational data from different geographic locations.

Information about nodes that comprise the Rivulet sketch is maintained as a prefix tree at each node. Addition and removal of nodes comprising the distributed sketch due to scaling maneuvers are propagated through the system and the prefix trees updated. Information within the prefix trees is designed to be eventually consistent. Alongside each vertex in this prefix tree, we maintain information about the spatiotemporal scope/range and the set of features available at the node.

Rivulet incorporates support for discrete and continuous queries specified by users. Queries specified by a user are evaluated over the distributed sketch. Depending on the spatial scopes that may be implicitly or explicitly specified within these queries, query evaluations will be performed on one or more nodes comprising the distributed sketch.

The entry point for these queries may be any of the nodes comprising the distributed sketch we refer to this node as the conduit for the particular query. During query evaluations, a first step is to identify the set of Rivulet nodes that must be targeted for query evaluations. The conduit node consults its prefix tree to identify nodes to contact based on the spatial, chronological, and particular features implied in the query. Conduit nodes are responsible for forwarding queries to the relevant nodes comprising the distributed sketch where the query must be evaluated. The conduit node also responds to the client with the list of nodes that will be responding to the query.

Results from query evaluations performed over the distributed sketch are streamed back to the client. Precisely what is streamed depends on the nature of the query. For example, if a client is interested in retrieving average temperatures in Colorado for July 2015 and if there are multiple nodes (within Rivulet) that hold relevant portions of this data; then the query must be forwarded to all these nodes. The results returned from each of these nodes would include <temperature, frequency> tuples that would then be combined at the client side to compute the final average temperature.

Alternatively, Rivulet nodes participating in the query evaluations will return vertices and edges (from their internal sketches) that satisfied the specified query constraints. The vertices and edges will be restricted to those pertinent to the query. For example, if a query targets a particular feature (say humidity) then vertices/edges relating to other features not targeted by the query will be pruned and not returned. Spatiotemporal features are exempt from this constraint. As vertices and edges are streamed back to the client from multiple nodes comprising Rivulet, these are organized into a graph that is then used to return the final results.

The prefix tree also allows query evaluations during both upscaling and downscaling operations. During query evaluations, queries are forwarded to child nodes (created during a recent upscaling operation). When the conduit forwards a query to a Rivulet node and the node is unavailable, it is either due to a failure at that node or a downscaling operation that merges the child node with its parent node. When a node is unreachable, the conduit forwards the query to the parent node.

4.6 Types of Queries

Queries within Rivulet can be discrete or continuous queries. Unlike discrete queries that are evaluated once, continuous queries are evaluated periodically or when observations become available. Though observations are never stored on stable storage, queries specified by users may target spatial and chronological scopes. In the case of continuous queries,

the chronological range implicitly evolves as new observations trickle in; in the case of windowing operators, the specified chronological window continually slides over the observational streams. We classify queries discrete or continuous supported by Rivulet into 5 broad queries. These include:

1. Filter queries that specify inequality/relational queries along one or more dimensions.
2. Statistical queries allow users to explore statistical properties of the observational space. 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 also of feature outliers based on Chebyshevs rule about the distribution of feature values.
3. Density queries support analysis over 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.
4. 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 repeated observations.
5. Inferential queries are intended to support projections about how the feature is expected to evolve over a spatiotemporal scope. In this study, these projections are predicated on the registration of time-series models at the particular scope. Specifically, we rely on exponential smoothing methods to make forecasts about the estimated values of features at different time points. Besides compact representations and faster evaluations that are amenable to keeping pace with high data arrival rates, smoothing methods are able to handle trends and seasonality that often underpin observational data. We rely on the Holt-Winters model. Here, the trend and seasonality components for the forecast are computed separately and updated independently based on the error computed from the forecasted and observed values. The seasonality component in the Holt-Winters model may either be additive or multiplicative; we work with the multiplicative components that are are known to perform better with real-world applications.

4.6.1 Filter Queries

4.6.2 Correlation/statistical queries

4.6.3 Kernel Density estimation queries

4.6.4 Set Queries

- Membership (Bloom Filters)
- HyperLogLog: probabilistic estimator for approximating the cardinality of a dataset
- Count-Min: approximate frequencies of repeated elements within a dataset

4.6.5 Forecasting Queries

Queries that we have discussed so far report on what has happened; a forecasting query requests information about what will happen.

4.7 Support for High throughput Query Evaluations

4.8 Coping with Failures in Rivulet

5. PERFORMANCE EVALUATION

5.1 Datasets and Experimental Setup

Our subject dataset was sourced from the NOAA North American Mesoscale (NAM) Forecast System [5]. 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. Rivulet was executed under the OpenJDK Java runtime, version 1.8.0_72.

5.2 Micro benchmarks

5.2.1 Dynamic Scaling

We evaluated how Rivulet dynamically scales when the data ingestion rate is varied. An artificial computation logic was used in place of regular Rivulet sketch update function to gain a better control over the throughtput of a Rivulet node. The state maintained at nodes were minimum, hence there was no significant memory pressure. The data ingestion rate was varied over time such that the peak data ingestion rate is less than the highest possible throughput which will create a backlog at Rivulet nodes. We used the number of sketch instances created in the system to quantify the scaling activities. If the system scales out, more sketch instances will be created in child nodes after the targeted load migration. We started with a single Rivulet node and allowed the system to dynamically scale. As it can be observed in Figure 9, the number of sketch instances vary with the ingestion rate. Since we allow aggressive scale out, it shows a rapid scale out activity with high data ingestion rates whereas scaling in takes place gradually with one sub region (hence one sketch) at a time.

5.3 System Benchmarks

6. RELATED WORK

Galileo [] 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

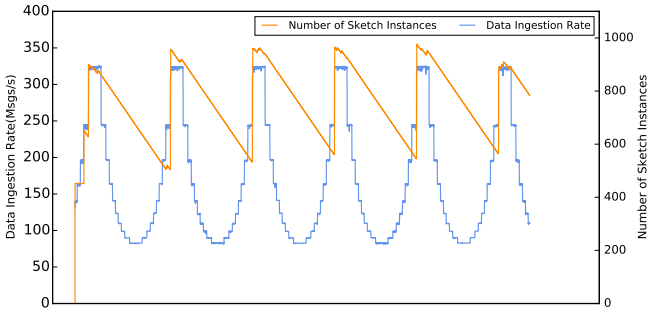

Figure 9: The variation of number of sketch instances with the data ingestion rate.

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.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a framework for constructing a distributed sketch over spatiotemporal observational streams. Rivulet maintains compact representation of the observational space, supports dynamic upscaling and downscaling to preserve responsiveness and avoid overprovisioning, and supports a rich set of queries to explore the observational space. Our methodology for achieving this is not restricted to the Neptune and is broadly applicable to other stream processing systems. Our empirical benchmarks, with real-world observational data, demonstrate the suitability of our approach.We now make a set of assertions pertaining to the research questions that we explored in this study.

RQ-1: Extracting metadata from observational data streams as they come in. We then check to see if the sketch needs to be updated. Each vertex represents a range of values; We achieve compactness because the number of vertices in the graph is influenced by the number of ranges and how they are organized both of which are dynamically managed in Rivulet. We also maintain summary statistics within these vertices to track the distribution/dispersion of feature values and the frequency with values fall within that range. Maintaining such summary statistics in an online fashion precludes the need to maintain fine-grained information that is memory intensive.

RQ-2: Data arrivals in observational settings are not uniform and the underlying infrastructure must be responsive to it. There will be variability in the rates and volumes of data arrivals from different geolocations due to differences and flux in the number of sensing sources. Rivulets scaling mechanism avoid overprovisioning via targeted scaling of portions of Rivulet. Targeted scaling allows us to alleviate situations where sketch updates cannot keep pace with data arrival rates. Memory pressure is also taken into account during upscaling, downscaling, and creation of spares. Accounting for memory pressure by tracking page faults, swap space, and available free memory allows us to cope with situations such as thrashing that may be induced by other collocated processes. Since Rivulet is memory-resident (to avoid disk I/O costs) tracking strains induced either by nodes comprising the Rivulet distributed sketch or the collocated processes is critical.

Given the rates of data arrivals, reactive schemes for scaling would result in updates and query evaluations over portions of the feature space to be slow. Our proactive scaling scheme triggers dynamic upscaling and downscaling based on the expected data arrival rates and the accompanying memory footprints. Our distributed locking scheme avoid deadlocks and liveness issues during scaling operations.

RQ-3: Rivulet is spatially explicit, with nodes comprising distributed sketch responsible for managing particular geospatial scopes. Scaling decisions are localized to particular nodes and involve load shedding during upscaling and fusion during downscaling. Each Rivulet node also maintains a prefix tree that maintains compact information about the nodes managing geospatial scopes. At each node, the metadata graph maintains information about different chronological time ranges; the system maintains fine-grained information about recent data and coarser grained information about older, historical data. This structure supports incremental scaling, allows nodes to effectively assimilate observations at high rates, and redirect queries to nodes where they should be evaluated. Our graph based organization of the extracted metadata allows us to support a rich set of queries without compromising on timeliness; the accuracy of these evaluations is influenced by the granularity of the representations.

RQ-4: Since we support targeted upscaling and downscaling based on the data arrival rates and memory pressure, the assimilations and query evaluations can keep pace with the observations. During query evaluations, only those Rivulet nodes that hold portions of the observational space implicitly or explicitly targeted by the query are involved in the query evaluations, and that to independently and without synchronizing with each other: this allows us to support high throughput query evaluations.

Our future work will target support for Rivulet to be used as input for long-running computations that are expressed as MapReduce jobs. Such jobs that would execute periodically and on potentially varying number of machines could target the entire observational space or only the most recent ones.

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).

8. REFERENCES

[1] T. Buddhika and S. Pallickara. Neptune: Real time stream processing for internet of things and sensing

environments. In *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium (To appear)*, 2016.

[2] M. Kristan, A. Leonardis, and D. Skoaj. Multivariate online kernel density estimation with gaussian kernels. *Pattern Recognition*, 44(1011):2630 – 2642, 2011. Semi-Supervised Learning for Visual Content Analysis and Understanding.

[3] 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.

[4] 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.

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

[6] G. Niemeyer. Geohash, 2008.

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

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