# Rubiks: Rapid Explorations and Summarization over High Dimensional Spatiotemporal Datasets

Saptashwa Mitra
Matt Young
sapmitra@colostate.com
asterix@colostate.com
Colorado State University
Fort Collins, Colorado, USA

Jay Breidt
breidt-jay@norc.org
NORC at the University of Chicago
USA

Sangmi Pallickara
Shrideep Pallickara
sangmi@colostate.com
shrideep@colostate.com
Colorado State University
Fort Collins, Colorado, USA

## ABSTRACT

Exponential growth in spatial data volumes have occurred alongside increases in the dimensionality of datasets and the rates at which observations are generated. Rapid summarization and explorations of such datasets are a precursor to several downstream operations including data wrangling, preprocessing, hypothesis formulation, and model construction among others. However, researchers are stymied both by the dimensionality and data volumes that often entail extensive data movements, computation overheads, and I/O. Here, we describe our methodology to support effective summarizations and explorations at scale over arbitrary spatiotemporal scopes, which encapsulate the spatial extents, temporal bounds, or combinations thereof over the data space of interest. Summarizations can be performed over all variables representing the dataspace or subsets specified by the user. We extend the concept of data cubes to encompass spatiotemporal datasets with high-dimensionality and where there might be significant gaps in the data because measurements (or observations) of diverse variables are not synchronized and may occur at diverse rates. We couple our data summarization features with a rapid Choropleth visualizer that allows users to explore spatial variations of diverse measures of interest. We validate these concepts in the context of an Environmental Protection Agency dataset which tracks over 4000 chemical pollutants, presenting in natural water sources across the United States from 1970 onwards.

## KEYWORDS

Spatial data, data cubes, summarizations

## 1 INTRODUCTION

Proliferation of data sources such as sensors, simulations, and scientific models have all contributed to growth in data volumes. This exponential growth in data volumes has continued uninterrupted over the past couple of decades. The crux of this study focuses on spatial data, where the data items are geocoded (data also have latitude and longitude coordinates associated with them). In many cases, chronological information in the form of timestamps representing when the observation was made are also included.

Spatial data occur naturally in several domains and their growth has been sustained by the ability to effectively and efficiently monitor spatiotemporally involving phenomena. Miniaturization and improvements in battery capacity alongside enhancements in the quality and capacity of networks have led to a proliferation of monitoring devices and their use in monitoring diverse phenomena. All of these have contributed to a sustained growth in the diversity and types of phenomena that are being monitored. The spatial extent to which these measurements can be attributed may be either point-based (geocoded using <lat,long> coordinates) or shape-based. In the case of shape-based geocodings the shapes may be expressed as N-sided polygon where each vertex is identified using <lat,long> coordinates. Spatiotemporally evolving phenomena are used to understand and inform decision-making in domains such as agriculture, geosciences, epidemiology, monitoring of environmental and ecological hazards, and commerce.

Rapid explorations of the dataspace are a precursor to data analysis, wrangling and visualization. Because such explorations allow the researcher to identify contours of the data space, they are critical to informing several aspects of downstream analyses. In the case of spatiotemporally evolving data, these explorations need to account for spatial and temporal dimensions in addition to the inherent multidimensional nature of the data. As such, these explorations should facilitate summarization across arbitrary spatiotemporal scopes. The term *spatiotemporal scope* refers to the scope of the data of interest. A user may be interested in specific spatial extents that may or may not be geographically contiguous. Similarly, the temporal bounds associated with the data of interest may vary from user to user. Finally, a user may be interested in a combination of constraints that are specified over the spatial and temporal dimensions.

Rapid summarizations of available data at arbitrary spatiotemporal scopes are critical to (1) informing data wrangling, (2) informing hypothesis formulation, (3) identifying broad brushstroke patterns in the data, and (4) identifying data to fit models over among others. A key requirement is that these summarizations are timely, scalable, and performant alongside the ability to perform these explorations at high throughput. The crux of this study to perform rapid summarizations and explorations of the spatiotemporal dataspace.

Data cubes[Gray et al. 1997] are considered an effective mechanism to facilitate such summarizations[Gray et al. 1997; Lins et al. 2013; Wang et al. 2023]: we extend this concept of data cubes to spatiotemporal data spaces where the number of observations may be very large. Crucially, we support these aggregations at scale, with low latency, alongside the ability to perform these operations along diverse spatial hierarchies (administrative, watersheds, quadtiles, etc.). We explore these ideas in the context of our research prototype, Rubiks.



*BDCAT '23, December 4–7, 2023, Taormina (Messina), Italy*
2023. ACM ISBN 979-8-4007-0473-4/23/12.
https://doi.org/10.1145/3632366.3632393

## 1.1 Challenges

Rapid summarization of high-dimensional data spaces over arbitrary spatiotemporal scopes faces several challenges. These include:

(1) Data volumes: The datasets we consider are voluminous. Inefficient schemes that rely on multiple sweeps of the data may exacerbate the I/O requirements.
(2) Dimensionality of the datasets: The datasets we consider are high-dimensional and users may be interested in summarization and exploration capabilities across all features. Our empirical benchmarks are performed over a dataset where over 4,000 chemical pollutants are being tracked.
(3) The aggregations and summarization can be performed at diverse spatiotemporal scopes i.e., users can specify arbitrary chronological bounds along spatial bounds. The spatial bounds can be based on administrative boundaries, watershed, climatic regions, quad tiles, etc.
(4) The spatiotemporal characteristics of the data preclude efficiency of solutions that rely exclusively on indexing and query evaluation efficiency.

## 1.2 Research Questions

Within the broader overarching goal of summarization and aggregation capabilities at scale, specific research questions that we explore include:

**RQ-1:** How can we preserve interactivity?
**RQ-2:** How can support effective summarizations across arbitrary spatiotemporal scopes?
**RQ-3:** How can we scale with increases in data volumes, new data sources coming online, and continual (or streaming) data generation?

## 1.3 Overview of Approach

We leverage a novel mix of algorithmic, statistical and systems approaches to facilitate real time data summarization at scale across user-specified spatiotemporal scopes. Our methodology places no constraints on the size of these spatiotemporal scopes.

We stage and disperse the data so that the data can be collated effectively without significant data movements. Data from moderately sized spatial extents (e.g., tracts, counties, or watershed boundaries) are collated and stored on the same machine.

Summarizations allow a researcher to spot patterns that arise at diverse spatiotemporal scopes. Summarizations provided by our data cubes include min, max, mean, median, variance, standard deviations, and distributional skew and kurtosis associated with individual features (or variables). We also supplement these measures by tracking the covariance across a set of user-specified features. Rubiks data cubes support pivot, aggregation, and disaggregation operations. Pivots allow the data cube to be probed across a specific dimension e.g., spatial, temporal, or any of the features encapsulated within the cube. The roll-up and drilldown operations relate to aggregation and disaggregation operations across spatiotemporal scopes. For example, a user may be interested in exploring the data space at coarse scales (roll-up) or at finer scales (drilldowns). These operations allow a user to specify interest over the dataspace at progressively larger or smaller spatial extents, time ranges, or combinations thereof.

Rather than compute these data cubes exhaustively every time a query is issued, we perform a limited number of one-time precomputations that we then leverage to support data cube operations. The smallest unit of data summarization in the system is a ***cubelet*** representing the smallest, indivisible spatiotemporal scope at which summarizations are performed. In our methodology, the scope associated with the cubelet is configurable. Data cubes are constructed from cubelets. Data cubes may either be constructed from cubelets or hierarchically constructed from other cubes. We leverage Welford's algorithm to compute the cubelets in an online, single-pass fashion. Information maintained within the data cubes are also amenable to leveraging the same online method to compute data cubes at ever coarser scales. Our methodology also allows data cubes to be constructed from non-contiguous spatiotemporal cubes.

A challenge that we also consider that the measurements across different variables are often not synchronized. For example, consider the case where multiple monitoring stations are profiling a water body for chemical pollutants. The pollutants may be measured at a different timepoints and frequencies. Correlation analysis over such measurements with non-concurrent sampling between the related attributes require special consideration and interpolation. For such irregular time-series of measurements, we implement kernel-based weighting in the computation of correlation and covariance in our cubelets.

Our pairwise covariance computations allow a user to first identify the pairs of covariances that are of interest. To reduce the number of pairwise covariances that need to be maintained in the data cube – $O(N^2)$ for $N$ variables – we allow users to identify the set of M covariances that are of interest. Alternatively, the covariates of interest may be domain-specific or computed dynamically by the system based on occurrence of variables in queries.

In Rubiks, cubelets are space-efficient and persistently stored since they are used in the computation of data cubes that may span diverse spatiotemporal scopes. Persistent storage of the cubelets also precludes duplicate computations alongside repeated sweeps of the data involving I/O. The data cubes, on the other hand, are ephemeral meaning they are garbage collected after a period of time.

Our summarization capabilities are backed by a distributed cache that serves two key purposes. First, the cache is used to store data cubes that have been calculated based on user-specified queries. We also store cubelets that were used to construct these data cubes; the rationale for this is that it is often the case that users are incrementally refining queries to customize the spatiotemporal scopes of interest. As such, cubelets that are part of a query have a higher likelihood of inclusion in the refinement queries. Second, the cache can reduce duplicate processing alongside any I/O that such refinements entail. Our distributed cache relies on a LRU (least recently used) caching scheme with an additional preference for storing cubelets rather than data cubes when the cache needs to evict elements during a cache miss.

We supplement the summarization feature with the ability to visualize these summarizations using a Choropleth map to render spatial variations of measures of interest. As such, the queries may be composed visually, and the roll-ups and drilldowns can be performed using slider bars. We allow dynamically constructed data cubes to be visualized using our Choropleth map service.

### 1.4 Paper Contributions

This study describes a framework for summarization over voluminous, high-dimensional spatiotemporal dataspaces. Specific contributions of our methodology include:

(1) A scalable framework that supports continuous assimilation of data, targeted I/O, and cache-residency schemes to minimize duplicate processing.
(2) Our summarizations can be performed in near real-time regardless of the spatiotemporal scopes involved. Except for the pairwise covariances where users configure variable pairs of interest, the other aspects of summarizations are available for all variables of interest.
(3) Our summarization schemes are backed by a distributed caching scheme that preferentially caches cubelets and data cubes to reduce disk access times and re-computation costs.
(4) Finally, our methodology places no restrictions on the storage frameworks that host the voluminous datasets.

**Translational Impacts:** Voluminous, high-dimensional spatiotemporal datasets continue to be made available in several domains such as agriculture, epidemiology, monitoring of environmental and ecological hazards, forest fire predictions, etc. The proposed effort allows users to quickly explore the data space to identify portions of the data space that are of interest.

### 1.5 Paper Organization

The remainder of this paper is organized as follows. Section 2 outlines related works, followed by the background in Section 3 that introduces the nature of the actions and spatiotemporal user queries. Section 4 describes the contruction of Rubiks cubelets and its in-memory data model. Section 5 details our in-memory data store and its role in fast query evaluation. Experimental setups, performance benchmarks, and analysis of results are outlined in Section 6. Finally, Section 7 outlines our conclusions, followed by acknowledgements.

## 2 RELATED WORK

Approaches to data summarization have been explored using sketching algorithms [Buddhika et al. 2021a, 2017, 2021b] that leverage probabilistic data structures. Rubiks does not leverage probabilistic data structures and produces aggregations that have lower uncertainties associated with them. Some approaches have explored the use of metadata graphs to support diverse queries [Malensek et al. 2015, 2017]; a disadvantage of these approaches is that the memory footprint can be substantial and involve traversals that may be prolong query evaluation times. Brokering systems leverage distributed data structures to perform such coarse grained matching [Pallickara and Fox 2004], but are unable to support finer-grained queries of the type Rubiks supports. Rubiks is well-suited for deployments in traditional cluster-based settings and grid settings [Fox et al. 2003, 2005a,b].

The popularity of spatiotemporal data has intensified the demand for efficient analysis and query processing techniques. The confluence of spatial and temporal dimensions, along with the availability of a wide range of recordable attributes in these data, thanks to the improvement in observational equipment, presents unique challenges in terms of storage, retrieval, and analytical processing. Traditional databases often struggle to efficiently manage these attributes, necessitating the development of specialized solutions.

The concept of data cubes has gained traction as a structured means of aggregating and analyzing multi-dimensional data. These are data structures constructed dynamically or through a prefetching scheme[Battle et al. 2016; Mitra et al. 2019, 2021b] in anticipation of data tiles being queried in the future to improve latency. Often, multivariate data cubes are maintained at various resolutions to enable comprehensive and flexible analysis of data across varying resolutions, providing a multi-dimensional analytical framework[Lins et al. 2013; Mitra et al. 2023, 2021a; Pahins et al. 2017; Santos et al. 2011; Tao et al. 2019].

Techniques involving spatial hashing and distributed systems have emerged to manage large-scale spatial data. These approaches enhance scalability and enable efficient query processing in distributed environments. Cache-based storage of data in memory in a distributed fashion has also proven to improve interactivity[Li et al. 2015, 2017, 2013; Pan et al. 2018; Paul and Fei 2001]. In dynamic datasets, the evolving nature of data requires adaptive analytical approaches. Existing solutions struggle to efficiently update and maintain analytical structures while accommodating continuous data changes.

The use of distributed clusters has become instrumental in handling large-scale data processing. Techniques such as DHT-based distribution and cluster synchronization have evolved to cater to the complex requirements of spatiotemporal data[Whitman et al. 2014]. Several existing frameworks address spatiotemporal data analysis, including GeoSpark[Yu et al. 2015], GeoMesa[Hughes et al. 2015], and STARK[Hagedorn et al. 2017].

While these frameworks offer valuable insights, Rubiks distinguishes itself through its cubelets-based approach and its specialized methods for handling evolving datasets. Additionally, algorithms that can update aggregations incrementally, like Welford's algorithm, have demonstrated their efficacy in handling real-time data changes while minimizing computational overhead[Rehfeld et al. 2011; Welford 1962]. The count-min (CM) sketch provides event frequencies using sublinear memory space, where an event could be a particular feature value or observation [Cormode 2009; Cormode and Muthukrishnan 2005]. The CM sketch is closely related to Bloom filters, which employ hash functions over a fixed-size bit array to determine set membership. With Bloom filters, false positives are possible but false negatives are not [Bloom 1970]. Several streaming algorithms have been developed to determine the number of distinct (unique) elements in a multiset, such as HyperLogLog++ [Heule et al. 2013], HyperLogLog, LogLog, and Linear Counting [Whang et al. 1990].Rubiks implements online update of statistics in its cubelets leading to improved speed of update.

Rubiks' novel framework improves spatiotemporal data analysis over varying resolutions by effectively addressing the complexities of evolving datasets. Its unique blend of cubelet aggregation, dynamic cubelet updates, distributed partitioning through spatiotemporal hashing, and distributed computing addresses a key need in the domain of large-scale spatiotemporal data analysis.

# 3 BACKGROUND

A data cube is a data structure that allows for efficient analysis of large volumes of data from multiple dimensions. It is commonly used in the field of business intelligence and data analytics.

A data cube organizes data in a multi-dimensional array format, where each dimension represents a different attribute or measure of the data. For example, in retail analytics, dimensions could include time, product, store location, and customer segment. The measures could include sales revenue, quantity sold, and profit.

Data cubes allow for efficient querying and analysis of data by providing pre-computed aggregations along each dimension. These aggregations, often referred to as "cuboids", provide various levels of granularity and summaries of the data. By pre-computing these aggregations, queries can be executed much faster compared to traditional relational database approaches.

In the context of distributed analytics, data cubes can be particularly beneficial. They enable distributed storage and processing of large datasets across multiple nodes or machines in a cluster. By partitioning the data cube and distributing its components across the cluster, each node can independently process a portion of the data, allowing for parallel and distributed analysis.

Distributed data cubes help in scaling analytics workloads by distributing the computation and storage resources across multiple machines. This approach allows for handling larger datasets, processing queries faster, and accommodating increased user concurrency. It can significantly improve the performance and scalability of analytics systems, enabling organizations to derive insights from vast amounts of data more efficiently.

Scaling data cubes to handle ever-increasing data volumes and user concurrency can be a challenge. Distributing the cube across multiple nodes or machines in a distributed environment can help address scalability concerns, but it introduces additional complexities in terms of data partitioning, synchronization, and load balancing.

# 4 METHODOLOGY

Our methodology facilitates construction of hierarchical datacubes. Data cubes may be constructed from cubelets (smallest, indivisible unit of aggregation in the system) or from other data cubes. Data cubes generate aggregated summarization of measurements over a spatiotemporal scope at varying levels of coarseness, based on their resolution. Here, we demonstrate our methodology for computing and analyzing data cubes over disjoint spatiotemporal extents.

## 4.1 Cubelets

In the Rubiks framework, a cubelet serves as the fundamental unit of aggregation and analysis. These cubelets play a pivotal role in encapsulating aggregated values across a diverse spectrum of measurements within a well-defined spatiotemporal scope. With these cubelets, we develop a dynamic analytical framework that facilitates iteratively identifying regions of interest that satisfy desired properties or covariences.

Our cubelets encapsulate statistical summaries such as counts, means, minimums, maximums, and standard deviations, alongside distributional skew and kurtosis, for all observations for a particular variable within the specified spatiotemporal extent. The data encapsulated within a cubelet is space-efficient and is amenable to aggregations i.e., cubelets can combined to produce a new data cube that encapsulates the aggregated measures of interest.

The ability to hierarchically aggregate cubelets (and cubes) facilitate a comprehensive exploration of spatiotemporal patterns, trends, and relationships across the entirety of the dataset's geographic and temporal domain. By orchestrating queries that target data cubes, *we aim to pinpoint regions of interest that align with specific criteria or exhibit correlated behaviors with a predefined set of features*, enabling targeted analysis of intricate spatial and temporal phenomena and extraction of nuanced insights, contributing to informed decision-making in a wide array of applications.

Cubelets are created over a configurable spatiotemporal scope. The cubelets can be aggregated hierarchically into data cubes at varying spatiotemporal resolutions. We describe the hierarchical organization of cubelets in section 4.6.

In Rubiks, cubelets are *perennial* while the data cubes are ephemeral. Cubelets are created and persisted on stable storage (and thus are perennial) along with actual data points during ingestion. These cubelets have a predetermined resolution and constitute the lowest level of the cube hierarchy. Cubes are coarser in the sense that they are computed on an on-demand basis from cubelets or other cubes in a hierarchical fashion and are ephemeral (i.e., they may or may not be persisted to disk).

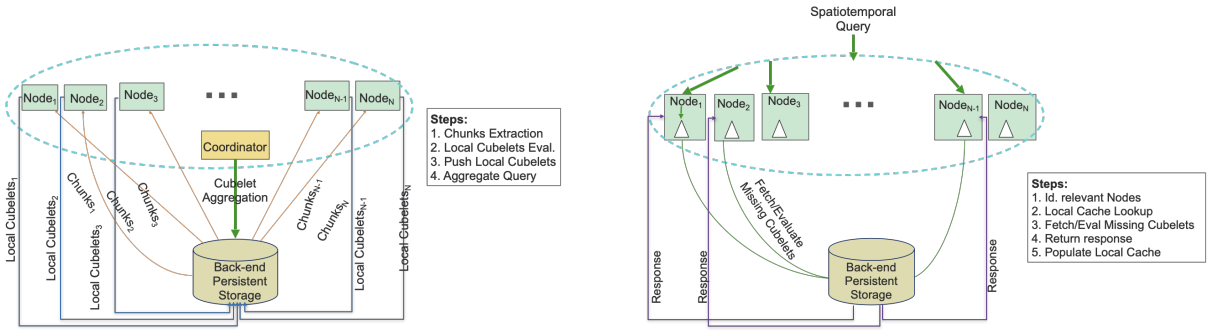
## 4.2 Cubelet Content

Cubelets summarize data from a particular spatial extent and are constructed from persistent data stored on disk (we place no constraints on the storage framework used to store such data). Each cubelet summarizes data for a configurable but system-wide spatiotemporal scope. The cubelet comprises a set of metadata attributes recorded within that region. The supported metadata includes essential statistical measures such as count, mean, minimum, maximum, and standard deviation for each attribute.

To enhance the analytical capabilities of cubelets, for a predefined set of attribute pairs, we also maintain running covariances within each cubelet. These covariances facilitate the evaluation of Pearson correlation coefficients at runtime, enabling researchers to gain insights into the relationships between different attributes within the cubelet.

## 4.3 Cubelet Spatiotemporal Bound

Rubiks offers the flexibility to construct data cubes at different spatiotemporal extents, tailored to the specific dataset, creating non-overlapping regions as the foundation for construction of data cubes. In Rubiks, we allow data cubes to be created over varying types of disjoint geospatial bounds, such as quadtiles, Hydrologic Unit Codes (HUC)[Seaber et al. 1987], and Federal Information Processing Standards (FIPS) codes that are used by the U.S. Census Bureau. This feature enables the analysis of data with diverse spatial characteristics, accommodating datasets that might have irregular or complex geographical boundaries. By supporting multiple geospatial bounds, Rubiks allows researchers to perform detailed analyses on localized regions while also gaining insights into broader geographic trends, fostering a more comprehensive




**(a) Cubelet construction during ingestion** **(b) Cubelet fetching/ dynamic evaluation during queries**

**Figure (1) Rubiks cubelet contruction and fetching**

exploration of spatiotemporal patterns and relationships within the data.

Perennial cubelets represent the finest level of aggregation and are persisted both on-disk over our distributed storage, as well as in-memory cache that we construct over the cluster nodes. These can be hierarchically combined to create coarser aggregates – the ephemeral cubes – facilitating a multi-resolution analysis of spatiotemporal patterns and trends. This provides a powerful tool for efficient and flexible exploration of large-scale point datasets with varying granularities. Ephemeral cubes are constructed as client-queries get evaluated server-side to enable collaborative query evaluation. At the finest level, perennial cubelets are constructed and updated during data ingestion by aggregating and summarizing point data that fall within a predefined spatiotemporal extent – for instance, over a spatial bound of a HUC12 boundary and temporal bound of a single day.

## 4.4 Distributed Ingestion: Perennial Cubelet Generation

Perennial cubelets are generated during data ingestion. In Fig. 1a, we illustrate the process of generating these cubelets. Preprocessing of incoming voluminous data in a standalone fashion can be time-consuming and compute-intensive. Rubiks relies on a distributed cluster of nodes for handling data ingestion, cubelet creation and query evaluation.

During ingestion, incoming data-points are partitioned into chunks and ingested in a distributed manner across our cluster nodes. Each node independently computes its local set of cubelets, contributing updates to a temporary set of cubelets in the distributed storage backend. Subsequently, a coordinator node initiates an aggregation query to combine local cubelets with overlapping keys, if any, into usable perennial cubelets. Only cubelets are constructed and persisted; data cubes themselves are constructed hierarchically on an on-demand basis.

## 4.5 Cubelet Update

To adapt to the continuous updates to the underlying storage, we ensure concurrent data ingestion and the update of cubelets in persistent memory.

*4.5.1 Welford's Algorithm for Rapid construction/Updates.* To ensure efficiency and scalability within Rubiks, we employ dynamic merging and updates of cubelets using Welford's algorithm, which provides a computationally efficient (single-pass) approach for incrementally calculating the mean and variance as new data are added or cubelets are merged. This method allows for real-time updates and analysis without the need to recompute the entire dataset, reducing both computational complexity and memory requirements.

Leveraging Welford's algorithm and associated metadata mean that our cubelets can efficiently accommodate data updates and adapt to changing input without sacrificing analytical accuracy. The algorithm's incremental, online nature makes it particularly well-suited for handling continuous data ingestion and maintaining up-to-date statistics within cubelets and across data cubes that are hierarchically constructed using cubelets and other data cubes. As a result, both cubelets and data cubes can dynamically adjust to new data points, supporting real-time analyses and ensuring a robust and scalable solution for data management and analysis. Utilizing Welford statistics for aggregation over cubelets allows us to 1) rapidly identify cubelets that require change/creation, and 2) perform rapid, decentralized updates over our cluster.

Due to the disjoint nature of measurements of attributes at a monitoring station, recorded measurements of any pair of desired attributes at the same location is never guaranteed to be concurrent. This complicates the process of measuring correlation between such attributes. To account for this situation, we aim to estimate the correlation measures in these cases by interpolating the recorded values based on how distant their measurements are in time. We explain the process of correlation computation for such misaligned measurements next.

*4.5.2 Correlation estimation for misaligned time series.* If two time series $x$ and $y$ are observed at irregular time points $\{s_i\}_{i=1}^{n_x} \neq \{t_j\}_{j=1}^{n_y}$, the empirical means $\widehat{\mu}_x, \widehat{\mu}_y$ and empirical standard deviations $\widehat{\sigma}_x, \widehat{\sigma}_y$ for the two series can be computed directly. The empirical correlation, however, can only be computed directly if the observation times are aligned ($n = n_x = n_y$, $s_1 = t_1, s_2 = t_2, \ldots, s_n = t_n$):

$$\widehat{\rho}_{xy} = \frac{1}{n-1} \sum_{j=1}^{n} \left\{ \frac{x(t_j) - \widehat{\mu}_x}{\widehat{\sigma}_x} \right\} \left\{ \frac{y(t_j) - \widehat{\mu}_y}{\widehat{\sigma}_y} \right\}.$$

If observation times for the two series are misaligned, we use the non-rectangular kernel approach described in [Rehfeld et al. 2011] to approximate the correlation. Let

$$K_h(s,t) = \frac{1}{h\sqrt{2\pi}} \exp\left\{\frac{-(s-t)^2}{2h^2}\right\}$$

denote the Gaussian kernel function with bandwidth parameter $h$. This kernel function is used to determine which time points between the $x$ and $y$ series are close enough to be used in estimating the correlation, via

$$\begin{aligned}\widetilde{\rho}_{xy} \quad = \quad & \frac{1}{\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} K_h(s_i - t_j)} \\ & \times\Bigg[\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} \frac{x(s_i)}{\widehat{\sigma}_x}\frac{y(t_j)}{\widehat{\sigma}_y} K_h(s_i - t_j) \\ & -\frac{\widehat{\mu}_y}{\widehat{\sigma}_y}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\frac{x(s_i)}{\widehat{\sigma}_x} K_h(s_i - t_j) \\ & -\frac{\widehat{\mu}_x}{\widehat{\sigma}_x}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\frac{y(t_j)}{\widehat{\sigma}_y} K_h(s_i - t_j) \\ & +\frac{\widehat{\mu}_x}{\widehat{\sigma}_x}\frac{\widehat{\mu}_y}{\widehat{\sigma}_y}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} K_h(s_i - t_j)\Bigg]. \end{aligned} \tag{1}$$

We compute the average distances $\Delta_s, \Delta_t$ between consecutive time points in $\{s_i\}_{i=1}^{n_x}$, $\{t_j\}_{j=1}^{n_y}$, respectively, and choose $h = 0.25 \times \max\{\Delta_s, \Delta_t\}$, following [Rehfeld et al. 2011]. As noted in [Rehfeld et al. 2011], $\widetilde{\rho}_{xy}$ is not guaranteed to lie within $[-1, 1]$; we set it equal to the closest boundary value if it falls outside.

If information from two cubelets is to be combined, let $\{s_i^{(k)}\}_{i=1}^{n_x^{(k)}}$ and $\{t_j^{(k)}\}_{j=1}^{n_y^{(k)}}$ denote the observation time points for cubelets $k = 1, 2$. Assume that from pilot analysis a single value of $h$ can be determined across cubelets. Further, assume that

$$K_h\left(s_i^{(1)}, t_j^{(2)}\right) \simeq 0, \quad K_h\left(s_i^{(2)}, t_j^{(1)}\right) \simeq 0;$$

that is, a misaligned pair in two different cubelets has time points sufficiently far apart to contribute nothing to the correlation computation. Then replace each cubelet mean and standard deviation in equation (1) by the combined mean and standard deviation; and replace each double sum in (1) by adding the two corresponding double sums (one for each cubelet); e.g., replace the first double sum in the numerator by

$$\sum_{k=1}^{2}\sum_{i=1}^{n_x^{(k)}}\sum_{j=1}^{n_y^{(k)}} \frac{x(s_i^{(k)})}{\widehat{\sigma}_x}\frac{y(t_j^{(k)})}{\widehat{\sigma}_y} K_h(s_i^{(k)} - t_j^{(k)}).$$

In addition to the information already required for updating the mean and standard deviation when combining cubelets, this correlation computation requires storing for each cubelet the four distinct double sums in (1).

*4.5.3 HashGrid for Updating Cubelets.* In Rubiks, the need for continuous cubelet updates to ensure query accuracy stems from the dynamic nature of the underlying data store. To effectively accommodate this evolving data landscape, concurrent data ingestion and cubelet updates within persistent memory are critical. This process is orchestrated through a hashgrid-driven approach, aimed at ensuring accuracy of constructed data cubes with the evolving dataset through the following steps:

**Binary Hierarchical Hashgrid:** Rubiks maintains a binary hierarchical hashgrid, wherein each element corresponds to a specific cube. This hashgrid serves as a reference to indicate whether a cube is up-to-date or requires updating due to changes in the underlying data.

**Coordinator-Initiated Updates:** During execution of the aggregation, the coordinator node also monitors and tracks cubes that have undergone modifications since the last update. The coordinator node updates the hashgrid based on the modifications detected. Each corresponding element in the hashgrid is updated to reflect the current status of its respective cube – indicating whether it is up-to-date or not.

**Hierarchical Update Propagation:** The hierarchical structure of the hashgrid streamlines the propagation of updates. The coordinator node can efficiently update higher-level hashgrid elements based on changes in the lower levels. This hierarchical mechanism ensures a streamlined and efficient update process.

**Cluster-Wide Synchronization:** Once the hashgrid is updated by the coordinator, this updated hashgrid is disseminated to all the cluster nodes. This push informs each node about the cube that are currently out-of-sync and cannot be used for query evaluation due to outdated information.

By leveraging this hashgrid-driven approach, Rubiks seamlessly incorporates continuous data updates into its cubes. This process ensures that the cubes remain relevant and accurate, enabling accurate and up-to-date query evaluations even over dynamic, continually-evolving datasets.

## 4.6 Hierarchical Aggregation of Cubelets

The computed cubelets, which represent fine-grained spatiotemporal aggregates, are systematically organized into a hierarchical structure. This hierarchical organization is achieved through the aggregation of lower-level cubelets that lie within the bounds of a given *parent* cubelet, ensuring efficient representation and management of the cubelets. Additionally, specific hierarchical structures such as quadtiles, Hydrologic Unit Codes (HUC), and Federal Information Processing Standards (FIPS) codes are employed to cater to diverse geospatial bounds. Temporally, we allow aggregation to be in units of days, weeks, months, or years.

To form coarser aggregates at higher levels of the hierarchy, cubelets are combined. This merging process allows the creation of larger aggregations, providing a multi-resolution perspective of the data (Fig. 2). Moreover, higher-level spatiotemporal extents are applied to encompass multiple cubes of varying types, further enhancing the versatility of the hierarchical framework. By employing these methods, the hierarchical organization enables more insightful analysis of spatiotemporal patterns and trends within the datacubes.

The cube hierarchy (with cubelets at the lowest level and dynamically, recursively constructed data cubes) is maintained in the form of a metadata graph. However, since we can deterministically and hierarchically aggregate based on spatiotemporal bounds, there is

no need to maintain actual links between the cubes themselves. We maintain these cubes as a set of hashmaps, grouped by their spatial and temporal keys, allowing targeted, efficient $O(1)$ retrievals.

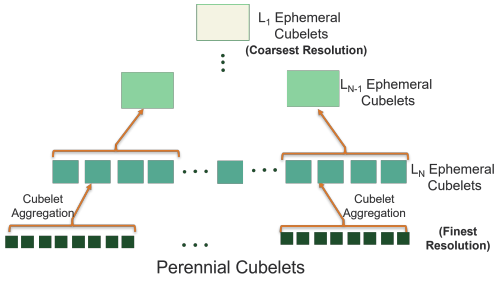



**Figure (2) Dynamic construction of cubelet hierarchy**

## 4.7 Query Evaluation

The following is a sample spatiotemporal query that we support at client-side. Rubiks supports spatiotemporal queries at varying levels of resolution (spatial_resolution, temporal_resolution) over any given viewport (Polygon) and timerange(Query_Time).

```
select pearson(iron,mercury), ...
from Aqua_Dataset
where coornidates in Polygon
and time_stamp in Query_Time
group by spatial_resolution, temporal_resolution
```

Fig. 1b provides an insight into the query evaluation process orchestrated by Rubiks. The system handles analytical queries over spatiotemporal data through the utilization of datacubes. We elaborate on the overall query evaluation mechanism of Rubiks in a distributed context. When a client query is initiated, it is initially directed towards the relevant cluster nodes (as explained further in the subsequent section). At each node, a search is conducted within the in-memory cache for cubes that either precisely match or can be repurposed to meet the requirements of the current query. Subsequently, the query is enhanced to retrieve any unfulfilled spatiotemporal extents from the backend storage.

In the backend storage, Rubiks engages in a search for ephemeral cubes that can be effectively employed or repurposed to furnish accurate responses for the ongoing query. For any cubes that are found missing, they are dynamically constructed from the collection of perennial cubelets and subsequently dispatched to the requesting node. In addition to addressing the query at hand, these newly formed data cubes are added to the roster of ephemeral cubes for future potential use. Data cubes created using this dynamic and targeted process are cached. By orchestrating this interplay of cache utilization, dynamic data cube construction, and query enhancements, Rubiks realizes a robust and responsive query evaluation mechanism that ensures efficient utilization of available data and computational resources. Crucially, correctness is preserved while ensuring efficient analysis of spatiotemporal datasets.

# 5 SYSTEM ARCHITECTURE

The Rubiks framework, along with its hierarchy of perennial and ephemeral cubelets, comprises a distributed query evaluation and graph-based caching system. We explain each of these components in detail.

## 5.1 Distributed cluster

We have designed a distributed data structure, Rubiks, that allows users to retrieve aggregated values over arbitrary spatiotemporal granularities over large-scale data collections. The data collections we consider comprise multidimensional observations that are stored in files - each observation has associated spatial coordinates (latitude and longitude) and an observational timestamp. Rubiks aggregates data based on the spatial coverage and temporal range using statistical data aggregation methods. The distributed nodes comprising Rubiks are organized as a distributed graph that maintains the level of granularity used for data aggregation. Rubiks' distribution across the cluster is orchestrated through a zero-hop Distributed Hash Table (DHT) architecture. The overall spatiotemporal scope of the data is uniformly partitioned among cluster nodes via a fusion of spatial and temporal hashing mechanisms, ensuring efficient identification of the responsible node(s) for any requested boundary. Depending on the specific dataset, we allow the spatial hashing to take various forms such as quadtiles, HUC12, or FIPS codes, involving a segment of the spatial code along with timestamps to distribute data across nodes. It's crucial to highlight that Rubiks' node architecture doesn't store raw data. Rather, it focuses on retaining cubelets that dynamically populate their in-memory cache over time. This approach optimizes resource usage by capitalizing on cubelets, which are compact and versatile analytical units. In essence, Rubiks cluster distribution, rooted in spatial and temporal hashing, and its cubelet-centric storage strategy collectively ensure efficient data organization and rapid retrieval, underscoring its ability to handle complex spatiotemporal queries with heightened efficacy.

## 5.2 In-memory Cache

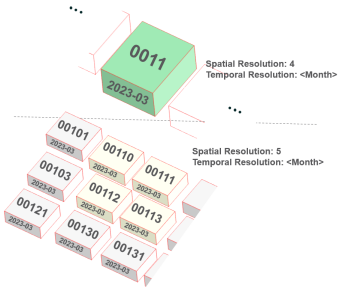



**Figure (3) Cubelet spatiotemporal bounds**

Rubiks' architecture includes a distributed cache on each node, organized to accommodate the spatiotemporal partitioning of the overall data domain. Cubelets pertinent to each node's assigned partition are maintained within their respective cache, establishing a sparse hierarchical metadata graph.

The in-memory cache structure within Rubiks diverges from conventional graph storage. Vertices correspond to sets of aggregated values sharing a common index key, while edges capture geospatial relations like granularities and proximity. Cached query

outputs and interim results optimize density based on granularity and spatiotemporal coverage. Rubiks' response time is nearly real-time as it navigates through its distributed hash tables, circumventing traditional graph traversal costs.

Rubiks optimizes response times by utilizing a distributed hash table architecture, replacing conventional graph storage structures. Each level, comprising sets of vertices with the same granularity, operates within a zero-hop DHT framework. These levels are navigated using a simplified array structure. This design significantly reduces computational complexity to O(1), enhancing Rubiks' efficiency and speed.

## 5.3 Dynamic Construction of Cubelet Hierarchy

Rubiks' hierarchical cubelets framework orchestrates the construction of coarser aggregates as part of the query evaluation process. This dynamic procedure enhances the versatility of cubelets, enabling their transformation into higher-level aggregations for multi-resolution analysis.

Initiated by query execution, the process commences with the retrieval of pertinent cubelets based on their spatiotemporal extents. Once retrieved, the system assesses the feasibility of merging neighboring cubelets to generate more comprehensive aggregations. Predefined rules guide this evaluation, ensuring that the resulting coarser cubelets align with desired parameters.

Upon identification of compatible cubelets, the system performs mathematical operations to fuse metadata attributes. These operations range from summing counts to updating various summary statistics including cross-feature covariances.

As queries progress, the hierarchical aggregation continually assembles coarser cubelets at higher levels. This adaptive approach integrates new aggregations into the query outcomes, affording users the flexibility to explore data at varying levels of detail. This allows Rubiks' to provide adaptable and efficient data exploration within its hierarchical cubelets framework.

## 5.4 In-Memory Storage and Cache Eviction Scheme

To ensure efficient storage and retrieval of cubelets within the hierarchical datacubes framework, we propose an in-memory storage scheme coupled with a cache eviction strategy to handle potential overflow. This section presents the design and implementation of this scheme.

*5.4.1 Cubelet Eviction and Freshness.* Rubiks leverages an eviction strategy that aligns with spatiotemporal user access patterns, prioritizing regions of interest over individual cubelets. Since the number of possible cubelets substantially outpaces the in-memory capacity, Rubiks allows configuration of thresholds for the count of cached cubelets.

Focusing beyond individual cubelet demand, the eviction strategy emphasizes **regions of interest**—frequently used spatiotemporal scopes at a specific instance. Rubiks' discerns relevance, identifies stale cubelets and swaps them for requested regions in case of memory pressure.

Rubiks uses the metric of **freshness** for this eviction strategy, calculated by multiplying cubelet access frequency with a time decay function, thus encapsulating both usage frequency and recency.

**Table (1) Cubelet Generation: Comparison between time (seconds) taken to create cubelets in a cold-start scenario vs daily updates**

| | County | Quadtiles | HUC-12 |
|---|---|---|---|
| **Cold-Start** | 187.53 | 219.90 | 363.77 |
| **Daily Updates** | 4.91 | 5.84 | 17.74 |

Cubelets are selected for replacement based on their freshness score. Fig. 3 provides a two-dimensional depiction of a spatiotemporal resolution with contained cubelets. Given recent access to regions $R_1$ and $R_2$, with spatiotemporal proximity implying future interest, our dispersion scheme extends a fraction of the freshness score to immediate neighborhood cubelets as well. This strategy safeguards against staleness in the immediate region, despite infrequent access.

The rationale for prioritizing regions over precise query extents is twofolds. First, users' queries often form a sequence, with queries spanning UI actions. Our methodology discerns spatiotemporal neighborhoods of interest encompassing potential future queries. Second, considering multiple users' queries, spatial access patterns cluster around small spatiotemporal areas. Thus, focusing on immediate query neighborhoods reconciles similar requests.

Cubelet replacement within Rubiks entails evicting stale cubelets with low-freshness scores till such time that memory pressure is relieved. This freshness-focused scheme ensures in-memory persistence of heavily accessed regions, enhancing query performance and latency.

## 5.5 Visualization of Cubelets

Cubelets allow exploratory data analysis over backend data such as heatmaps, time series plots, and interactive visualizations to identify patterns and trends across different levels of the cubelet hierarchy. We have used cubelets to generate heatmaps of water contaminant proliferation at the county, watershed boundary, and individual water body scales across the continental United States. We used a JavaScript front-end leveraging the DeckGL mapping framework to visualize heatmaps.

# 6 SYSTEM EVALUATION

## 6.1 Experimental Setup

To evaluate compute-intensive spatiotemporal queries and ingestion rates over our system, we profiled Rubiks over a cluster of 20 nodes. The data ingestion and query evaluation occurs in a spatiotemporally partitioned manner over the same cluster of 20 nodes. Each node in our distributed cluster is an Intel Xeon E5-2620v3, with 64 GB RAM, each with a Quadro P2200 GPU (5GB of memory) with 1280 cores and several local 7200RPM SATA hard disks. The data ingestion operations and spatiotemporal queries over the cluster get partitioned throughout the cluster uniformly based on the first 6 characters of their Quadtile key[qua 2018]. A sharded, replicated MongoDB cluster of 50 nodes was set up as our persistent storage that houses both raw data nd cubelets. The machines were organized into the following configurations: (1) 5 machines with mongos routers (2) 39 machines running mongod instances, co-located (3) 3 machines dedicated to running a Mongo config replica set.

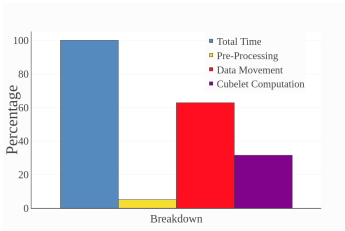



**Figure (4) Breakdown of overall cubelet construction time. From left to right: total time (blue), pre-processing (yellow), data movement (red), and cubelet computation (purple).**

## 6.2 Dataset and Spatiotemporal Extent

In order to test our framework against large-scale data, we use the Environmental Protection Agency's water quality dataset[epa 2023], which comprises over 4,000 different types of water quality-related measurements recorded for the vast majority of water bodiues in the United States from January 1st, 1970 through today. The dataset includes measurements from over 991,000 monitoring stations with a total of over 226 million data points, with new data ingested daily. Spatiotemporal queries over a dataset this voluminous present a significant challenge to query evaluation time, which the RUBIKS framework manages elegantly.

## 6.3 Cubelet Construction Time

We evaluate the time taken to construct the *perennial cubelets* over Rubiks in a cold-start scenario, where we have to ingest $\sim 226M$ entries into our distrubuted storage. Table 1 compares the time taken to construct cubelets constructed over varying non-overlapping geospatial bounds. We can see compared to the total number of records being ingested, the overall time to construct cubelets is quite low, in the order of a few minutes. Additionally, we note that the overall time taken to construct the cubelets is directly proportional to the total number of cubelets being constructed. For instance, for the total geospatial extent of the CONUS, the number of unique counties is 3163, the total number of quadtiles within the bounds is $\sim$ 15,000, whereas the total number of HUC-12 regions is ~87,000, which directly impacts the total number of cubelets required to be created and saved into our framework. As expected, the overall cubelet construction time is proof of that and it is to be noted that ideally, cubelet construction would occur alongside data ingestion.

We can also see that the update time for cubelets for incoming daily measurements is significantly low compared to a cold-start scenario. The difference in times taken for various geospatial cubelet bounds is also reflected here, as in the case of the cold-start scenario. We also profile a breakdown of the overall operation of cubelet construction during ingestion. During cold-start generation of perennial cubelets, we need to fetch the relevant distributed data to each computing node, perform preprocessing to load shapefiles to identify specific geospatial cubelet boundaries and perform aggregation and computation of cubelets, followed by persisting them. Fig. 4 demonstrates the percentage of overall cubelet generation time for a cold-start scenario where we create county-wise cubelets for each month over records starting from 1970 till current day.

**Table (2) Spatiotemporal Query: Comparison between latency (seconds) for varying sizes**

| | 1 Month | 1 Year | 10 Years |
|---|---|---|---|
| **Rubiks** | 3.3 | 3.32 | 6.51 |
| **Brute Force** | 12556.6 | 12606.2 | 12686.9 |

We note that data movements, which constitutes both moving ingested data to the cluster nodes and computed cubelets back to the persistent storage takes up a majority of the overall time.

## 6.4 Comparison of Accuracy

Since the Rubiks cubelets form the backbone of its query evaluations, we compare the accuracy of Rubiks' aggregate statistics against those computed through brute force. Since we use Welford's online algorithm to compute and update our cubelets, we expect them to accurately represent independent statistical measures such as mean, standard deviations, skewness and kurtosis, which are derived from the first four order of moments.

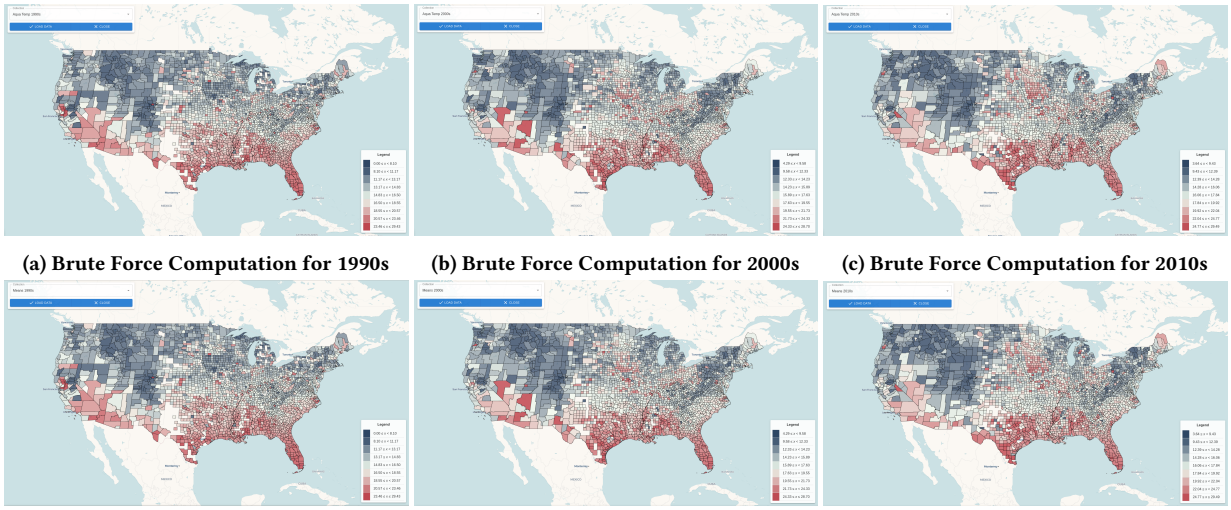
Fig. 5 depicts choropleth maps computed using both brute-force and through aggregation of Rubiks' perennial cubelets. The geospatial bounds used here are US counties. We compare the mean for the decades 1990s, 2000s, and 2010s for the water temperature in Celsius. We can see that along with significantly improved fetch latency the cubelets are constructed accurately (i.e., no deviations observed versus brute force traditional calculations) in an online fashion.

## 6.5 Query Evaluation Latency

We profile the improvement in latency through our Rubiks framework, compared to that of a spatiotemporal query over raw data. Here, we profile the latency over queries of varying size. We evaluate the time taken to compute county-wise aggregate statistics per month. By keeping the spatial bounds of the query fixed to the entire CONUS, we vary the temporal extent of the query to a month, a year and a decade. Table 2 profiles the average time taken for each of these 3 types of queries with and without the use of Rubiks cubelets. **We can see significant improvement in query times compared to fetching of raw data, with improvement ranging from ~3800-2000x.**

## 6.6 Improvement in Latency Through Caching

Rubiks' hierarchical distributed caching scheme at the cluster nodes helps avoid redundant processing and network communication. We evaluate the improvement in latency of various levels of overlapping queries. To profile the efficacy of our caching scheme, we populate our in-memory cache with 25, 50, and 100% of the query domain and execute random queries of a fixed size (CONUS, monthly) over that fixed domain. The average latency results for each of the 3 cases are shown in Table 3. We can see that the reduction in query latency is directy proportional to the amount of potential cache-hits which helps reduce the amount of cubelets that would need to be queried and fetched from the persistent store.



(a) Brute Force Computation for 1990s (b) Brute Force Computation for 2000s (c) Brute Force Computation for 2010s

(d) Cubelet-based Computation for 1990s (e) Cubelet-based Computation for 2000s (f) Cubelet-based Computation for 2010s

**Figure (5) Comparing choropleth maps vertically. The top row was generated using raw data, and the bottom row was generated using data cubes. Each choropleth map visualizes mean water temperature measured in degrees celcius, aggregated across counties. RUBIKS queries return accurate (i.e. identical to brute force) results but do so substantially (2000x - 3800x) faster.**

**Table (3) Query Latency: Comparison between query time (seconds) with varying levels of cache-hit**

| | 25% | 50% | 100% |
|---|---|---|---|
| **Latency (secs)** | 2.8 | 2.47 | 0.86 |

## 7 CONCLUSION

Here we described, RUBIKS, our framework for rapid summarization and explorations of high-dimensional, voluminous spatiotemporal datasets.

**RQ-1:** Preservation of interactivity is predicated on precomputing cubelets (atomic units) that can be leveraged in the computation of data cubes. Space efficiency of the cubelets alongside storage of additional metadata allows the same data cube to be leveraged in the computation of diverse data cubes. Relying on hierarchical spatial aggregations allows the operations to be targeted while also limiting the number of I/O operations that need to be performed. Our distributed caching schemes reduce duplicate processing and by prioritizing the residency of cubelets over ephemeral cubes reduce I/O requirements.

**RQ-2:** Hierarchical aggregations alongside the online Welford's algorithm lay the groundwork for effective summarizations across diverse spatiotemporal scopes. We supplement this with a kernel-based weighing of misaligned measurements to cope with the complexity of covariance computations when the measurements across variables are not synchronized in time and when the number of discrete measurements across variables are different. Deterministic identification of spatial scopes for aggregation (based on hierarchical prefix matching) alongside identification of temporal bounds allow us to be very targeted in the cubelets that are involved the calculation of data cube. We support diverse spatial extents: schemes that we currently support include administrative boundaries, watershed boundaries, and quad tiles.

**RQ-3:** Summarizations at scale are predicated on minimizing duplicate processing, leveraging hierarchical aggregations, and a distributed cache. The incremental creation of cubelets allows the framework to cope with continuous data arrivals and targeted updates. Persistent cubelets preclude recomputations while their space efficiency allows cache residency of a large number of cubelets to reduce disk I/O.

## ACKNOWLEDGMENTS

This research was supported by the National Science Foundation (OAC-1931363, CNS-2312319), and an NSF/NIFA Artificial Intelligence (AI) Institutes AI-CLIMATE Award [2023-03616].

## REFERENCES

2018. QuadTiles. https://wiki.openstreetmap.org/wiki/QuadTiles

2023. *Water Quality Data Download | US EPA.* https://www.epa.gov/waterdata/water-quality-data-download.

Leilani Battle, Remco Chang, and Michael Stonebraker. 2016. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data.* ACM, 1363–1375.

Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.

Thilina Buddhika, Matthew Malensek, Shrideep Pallickara, and Sangmi Lee Pallickara. 2021a. Living on the edge: Data transmission, storage, and analytics in continuous sensing environments. *ACM Transactions on Internet of Things* 2, 3 (2021), 1–31.

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

Thilina Buddhika, Sangmi Lee Pallickara, and Shrideep Pallickara. 2021b. Pebbles: Leveraging sketches for processing voluminous, high velocity data streams. *IEEE Transactions on Parallel and Distributed Systems* 32, 8 (2021), 2005–2020.

Graham Cormode. 2009. Count-Min Sketch.

Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

GC Fox, Hasan Bulut, Kangseok Kim, Sung-Hoon Ko, Sangmi Lee, Sangyoon Oh, Shrideep Pallickara, Xiaohong Qiu, Ahmet Uyar, Minjun Wang, et al. 2003. Collaborative web services and peer-to-peer Grids. *SIMULATION SERIES* 35, 1 (2003), 3–12.

Geoffrey Fox, Sang Lim, Shrideep Pallickara, and Marlon Pierce. 2005a. Message-based cellular peer-to-peer grids: foundations for secure federation and autonomic services. *Future Generation Computer Systems* 21, 3 (2005), 401–415.

Geoffrey Fox, Shrideep Pallickara, and Xi Rao. 2005b. Towards enabling peer-to-peer Grids. *Concurrency and Computation: Practice and Experience* 17, 7-8 (2005), 1109–1131.

Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery* 1 (1997), 29–53.

Stefan Hagedorn, Philipp Gotze, and Kai-Uwe Sattler. 2017. The STARK framework for spatio-temporal data analytics on spark. *Datenbanksysteme für Business, Technologie und Web (BTW 2017)* (2017).

Stefan Heule, Marc Nunkesser, and Alexander Hall. 2013. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*. 683–692.

James N Hughes, Andrew Annex, Christopher N Eichelberger, Anthony Fox, Andrew Hulbert, and Michael Ronquest. 2015. Geomesa: a distributed architecture for spatio-temporal fusion. In *Geospatial informatics, fusion, and motion video analytics V*, Vol. 9473. SPIE, 128–140.

Rui Li, Jiapei Fan, Xinxing Wang, Zhen Zhou, and Huayi Wu. 2015. Distributed cache replacement method for geospatial data using spatiotemporal locality-based sequence. *Geo-spatial Information Science* 18, 4 (2015), 171–182.

Rui Li, Wei Feng, Huayi Wu, and Qunying Huang. 2017. A replication strategy for a distributed high-speed caching system based on spatiotemporal access patterns of geospatial data. *Computers, Environment and Urban Systems* 61 (2017), 163–171.

Rui Li, Yinfeng Zhang, Zhengquan Xu, and Huayi Wu. 2013. A Load-balancing method for network GISs in a heterogeneous cluster-based system using access density. *Future Generation Computer Systems* 29, 2 (2013), 528–535.

Lauro Lins, James T Klosowski, and Carlos Scheidegger. 2013. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2456–2465.

Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. 2015. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing* 5, 1 (2015), 28–42.

Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. 2017. Hermes: Federating fog and cloud domains to support query evaluations in continuous sensing environments. *IEEE Cloud Computing* 4, 2 (2017), 54–62.

Saptashwa Mitra, Paahuni Khandelwal, Shrideep Pallickara, and Sangmi Lee Pallickara. 2019. Stash: Fast hierarchical aggregation queries for effective visual spatiotemporal explorations. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 1–11.

Saptashwa Mitra, Paahuni Khandelwal, Shrideep Pallickara, and Sangmi Lee Pallickara. 2023. ARGUS: Rapid Wildfire Tracking Using Satellite Data Collections. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE, 72–83.

Saptashwa Mitra, Daniel Rammer, Shrideep Pallickara, and Sangmi Lee Pallickara. 2021a. A generative approach to visualizing satellite data. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 815–816.

Saptashwa Mitra, Daniel Rammer, Shrideep Pallickara, and Sangmi Lee Pallickara. 2021b. Glance: A generative approach to interactive visualization of voluminous satellite imagery. In *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 359–367.

Cicero AL Pahins, Sean A Stephens, Carlos Scheidegger, and Joao LD Comba. 2017. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE transactions on visualization and computer graphics* 23, 1 (2017), 671–680.

Shrideep Pallickara and Geoffrey C Fox. 2004. On the matching of events in distributed brokering systems.. In *ITCC (2)*. 68–76.

Shaoming Pan, Lian Xiong, Zhengquan Xu, Yanwen Chong, and Qingxiang Meng. 2018. A dynamic replication management strategy in distributed GIS. *Computers & geosciences* 112 (2018), 1–8.

Sanjoy Paul and Zongming Fei. 2001. Distributed caching with centralized control. *Computer Communications* 24, 2 (2001), 256–268.

Kira Rehfeld, Norbert Marwan, Jobst Heitzig, and Jürgen Kurths. 2011. Comparison of correlation analysis techniques for irregularly sampled time series. *Nonlinear Processes in Geophysics* 18, 3 (2011), 389–404.

Luís Santos, João Coutinho-Rodrigues, and Carlos Henggeler Antunes. 2011. A web spatial decision support system for vehicle routing using Google Maps. *Decision Support Systems* 51, 1 (2011), 1–9.

Paul R Seaber, F Paul Kapinos, and George L Knapp. 1987. *Hydrologic unit maps*. Vol. 2294. US Government Printing Office Washington, DC, USA.

Wenbo Tao, Xiaoyu Liu, Çagatay Demiralp, Remco Chang, and Michael Stonebraker. 2019. Kyrix: Interactive visual data exploration at scale. CIDR.

Guoren Wang, Yue Zeng, Rong-Hua Li, Hongchao Qin, Xuanhua Shi, Yubin Xia, Xuequn Shang, and Liang Hong. 2023. Temporal Graph Cube. *IEEE Transactions on Knowledge and Data Engineering* (2023).

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

Kyu-Young Whang, Brad T Vander-Zanden, and Howard M Taylor. 1990. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems (TODS)* 15, 2 (1990), 208–229.

Randall T Whitman, Michael B Park, Sarah M Ambrose, and Erik G Hoel. 2014. Spatial indexing and analytics on Hadoop. In *Proceedings of the 22nd ACM SIGSPATIAL international conference on advances in geographic information systems*. 73–82.

Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2015. Geospark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems*. 1–4.