STASH : Fast Hierarchical Aggregation Queries for Effective Visual Spatiotemporal Explorations

Saptashwa Mitra
Department of Computer Science
Colorado State University
Fort Collins, USA
sapmitra@cs.colostate.edu

Paahuni Khandelwal
Department of Computer Science
Colorado State University
Fort Collins, USA
paahuni@cs.colostate.edu

Shrideep Pallickara
Department of Computer Science
Colorado State University
Fort Collins, USA
shrideep@cs.colostate.edu

Sangmi Lee Pallickara
Department of Computer Science
Colorado State University
Fort Collins, USA
sangmi@cs.colostate.edu

***Abstract*—The proliferation of sensors and observational instruments enable scientists to explore natural, spatiotemporal phenomena via explorative analysis and advanced modeling. Geospatial visualization, in particular, is an intuitive tool to identify patterns, enhance understanding of the data, and plan for subsequent analysis. However, seamless interactions between end-user devices and the sheer volume of data have been a challenge due to the limited bandwidth and data access latencies.**

In this paper, we introduce STASH, a distributed, in-memory cache for hierarchical aggregation and query evaluations. STASH is a middleware which can be loaded on top of a distributed file system. Users perform queries from a lightweight visualization interface at the front-end and the evaluations occur over the back-end storage system housing the raw data over which summarization and subsequent visualizations are to be performed. STASH facilitates fast exploratory analytics by caching relevant past query results based on their frequency and freshness to assist similar, future queries and avoid expensive disk I/O and network usage, thus reducing their latency. Additionally, STASH handles any hotspot that might result from a spike in user requests due to the spatial and temporal locality of their access patterns.

Our empirical benchmarks show that a STASH-enabled system reduces query latency of a basic system by over 5-folds and brings it down to interactive speed even for large country-sized spatiotemporal queries. We have contrasted STASH with existing cache-enabled analytics engines, such as ElasticSearch, and found that our STASH-enabled system reduced the aggregation query latency up to $\sim$70%. STASH also alleviated skewed workloads through its dynamic replication scheme and improved throughput by $\sim$40% in hotspot scenarios.

***Index Terms*—in-memory storage, distributed caching, aggregation query, visual analytics, exploratory analytics**

This research was supported by the Advanced Research Projects Agency-Energy(ARPA-E), the US National Science Foundation [ACI-1553685, OAC-1931363], the US Department of Homeland Security [D15PC00279], and a Cochran Family Professorship.

978-1-7281-4734-5/19/$31.00 ©2019 IEEE

I. Introduction

Interactive data visualization is a desirable feature for understanding phenomena. This is especially the case with voluminous georeferenced data harvested using IoT sensors and observational networks. Visualization allows scientists to explore the data at "rates resonant with the pace of human thought" [13], [14]. Scientists and analysts explore datasets using interactive visualization tools to identify patterns, formulate hypotheses, and determine the scope for refinements and subsequent analysis [31].

Within the visualization software, a user's interactions are transformed into a series of operations such as data retrieval, processing, and rendering. The visual analytics software (e.g., Tableau [6], QlikView [12], SAP Lumira [5], etc.) generates a set of queries based on the user's navigation; which are then issued to the backend data storage system such as a database or networked file system. For extensive scalability, options for cloud-based storage systems [1], [27] are often provided. However, since the query orchestration in the existing approach relies on the individual visualization software, those scopes are limited only to a single user. Moreover, the backend storage system does not cope with any distinctive characteristics of the visual exploration over the large-scale geospatial dataset.

There are unique challenges to supporting query evaluations for large-scale explorative visualization. First, at the scales that we consider, the number of data points that match a user's area of interest often exceeds the perceptual scalability limit. In particular, the resolutions of the resulting image is so high that an individual pixel cannot be seen and unlikely to be beneficial [34]. Therefore, an effective and flexible aggregation scheme is critical. Second, a query is often related to subsequent queries and they share properties (e.g. area, and attributes). Also, many users frequently navigate similar area especially if there are any national disasters or events. Both the frequency and trajectory of queries should be considered during the orchestration of query evaluations.

In this paper, we present STASH, a distributed in-memory data structure that supports hierarchical aggregation queries for the large-scale spatiotemporal visual exploration. Unlike existing backend storage systems, STASH works with underlying distributed file systems and provides an in-memory data storage layer that can flexibly scale based on available resources. STASH supports aggregation queries based on the user's navigation patterns such as slicing, dicing, zooming, panning, rolling-up and drilling-down and caches the aggregated results in main memory to provide reusability of the query outputs. STASH also alleviates data retrieval hotspots caused by popular areas-of-interest through a dynamic load-balancing scheme.

A. Research Questions

As part of this study, we explore the following research questions:
RQ-1: How can we facilitate low-latency, multi-level data aggregations over voluminous datasets?
RQ-2 How can we take a user's visual navigations into consideration to improve query evaluations?
RQ-3 How can the system cope with skews in access patterns (e.g. some geographical extents may be more heavily accessed than others) while preserving latencies and assorted functionalities?

B. Overview of Approach: Distributed cluster

In this study, we describe our methodology for effective aggregation queries to support interactive visualization applications. We have designed a distributed data structure, STASH, that allows users to cache, query and retrieve aggregated values with multiple granularities over large scale data collections. The data collections we consider comprise multidimensional observations that are stored in files - each observation has spatial coordinates (latitude and longitude) and an observational timestamp associated with it. STASH aggregates data based on the spatial coverage and temporal range using statistical data aggregation methods. The vertices comprising the STASH are organized as a distributed graph that maintains the level of resolutions of the granularity used for data aggregation.

Each vertex within STASH stores a set of aggregated values that share the same index key. Edges within STASH maintain properties indicating the geospatial relations between other vertices such as granularities or geospatial proximity. STASH collectively caches query outputs from users' queries. Therefore, STASH may have varying density based on granularity and spatiotemporal coverage. The system evaluates subsequent queries over the cached data points and accesses underlying file system only when the relevant portion is not available in STASH.

To achieve near real-time latency for data discovery, STASH maintains a map of distributed hash tables instead of a conventional graph storage system. Each set of vertices with the same granularity is dispersed over the cluster using a zero-hop DHT. We use a simple map to navigate between the different resolutions. This approach reduces the cost for data discovery to a single lookup over the local map and a single lookup over the DHT making the computational complexity O(1).

We have designed and implemented the query orchestration scheme based on common data access patterns derived from popular visual navigation features such as slicing, dicing, panning, zooming, drilling-down, and rolling-up. STASH uses these patterns to prioritize caching/pruning the aggregated data. Finally, STASH provides a dynamic load balancing scheme over the in-memory distributed data structure to manage high-throughput query evaluation.

On evaluation, our STASH-enabled system was found to reduce query latency of a basic system by over 5-folds for large spatiotemporal queries. Our system also reduced the aggregation query latency up to $\sim$70% compared to Elasticsearch (another caching-enabled analytics engine). STASH also alleviates skewed workloads through its dynamic replication scheme and improves throughput by $\sim$40% in hotspot scenarios.

C. Paper Contributions

Our methodology enables effective data retrievals that are aligned with the needs of visual exploration to provide real-time interactivity. In particular, our contributions include:
1) A distributed, dynamic caching scheme to alleviate I/O overheads associated with disk accesses. Query outputs and intermediate data are stored in our in-memory data structure and reused during subsequent query evaluations.
2) Support for rapid data discovery using a DHT map that minimizes query-forwarding between the storage nodes and local traversals within the data structure.
3) A dynamic data caching and query orchestration scheme that accounts for users' visual navigational patterns, especially those prevalent in dominant geospatial explorations. By improving data reusability during query evaluations we make frugal use of resources allowing us to do more with less.
4) A high-throughput visual query framework backed by a shared, in-memory caching scheme and dynamic load balancing mechanism that scales to a large number of users.

D. Paper Organization

The remainder of this paper is organized as follows. Section 2 outlines the background, introducing the nature of the actions and spatiotemporal user queries, followed by related works in section 3. Section 4 provides a discussion of STASH's data model and its role in fast query evaluation. Section 5 details STASH's leveraging of common visualization patterns and section 6 outlines the system architecture. Section 7 outlines the replication strategy for skewed workloads. Experimental setups, performance benchmarks, and analysis of results are outlined in Section 8. Finally, our conclusions and future research directions are described in Section 9.

II. BACKGROUND

A. Exploratory Visual Analytics

Visual exploration of voluminous spatiotemporal data involves multiple steps of data processing and numerous interactions between users and the system. To cope with the high density and data volume, commonly performed operations during explorations include navigating between segments of dataset and/or adjusting resolutions [31]. The user's interaction is translated into a series of requests to the backend data server by the front-end client application.

The front-end of the visual analytics system is usually a light-weight user interface with two key tasks: (1) translate a user action (e.g. panning, dicing, etc.) into a query over the data storage system and (2) processing the server response to extract a visual representation of the summary statistics (e.g., heatmap or histogram) over the visualization interface.

The back-end storage system processes a large number of requests from the users to provide values that the front-end system will render. For each request, the back-end system has to identify the segment of the dataset relevant to the query, aggregate over that segment at the desired resolution, and return a set of values over the query's spatiotemporal bounds.

B. Multi-Resolution Data Aggregation for the Spatiotemporal Visual Exploration

The simple data aggregation query provides a summary of the values that match the query. In contrast, the aggregation query for visual applications generates a set of pixel-level aggregations that matches the user's query. For example, the following SQL query shows an aggregation query for rendering maximum temperature values at a given spatial and temporal resolution(*spatial_resolution, temporal_resolution*) over an area and during a period specified in the *Query_Polygon* and *Query_Time*, respectively.

```
select max(temperature), ...
from NAM_Dataset
where coornidates in Query_Polygon
and time_stamp in Query_Time
group by spatial_resolution , temporal_resolution
```

The values within a minimum bounding box will be aggregated based on the temporal range specified by the *temporal_resolution* and the spatial range specified by *spatial_resolution*. These aggregations will be performed only over the data records matching the query. If a user tries to investigate an area with different resolutions, the query and resolutions should be modified. Therefore, the back-end data system should evaluate different aggregation queries for almost every user interaction.

III. Related Works

Several frameworks have been designed to support scalable visual analytics. Forecache [7] proposes a prefetching scheme that predicts data-tiles to be queried in the future based on the user's past behavior and movements to improve latency. Similar work involving pre-computations have been suggested in [19], [20], [23], [26], [30]. In imMens [20], multivariate data tiles are precomputed to provide scalable panning and zooming like in Google Maps [26]. [19] uses a data cube structure which stores all possible precomputed aggregations at multiple levels of resolutions over the database. Redis [4] an in-memory data storage system provides support for geospatial data analysis by indexing them as a sorted-set, sorted by geohashes. This ordered data-structure helps in faster indexing. However, the above systems do not scale with dataset size as they house the data structure in-memory. HashedCubes [23] is built on a pivot/array scheme that maintains a partial ordering scheme for all possible dimensions. It is more memory efficient than Nanocubes [19] since it avoids precomputations and uses sorted arrays to compute aggregations on-the-fly which, but, leads to higher query latency. Another system used for fast data visualization is Tableau [32], which can connect to a variety of underlying databases and support spatiotemporal queries over tabular data [10]. Bitmap indexing [29] store dataset as compressed bitmaps to leverage high-speed bit-wise operations. However, the increase of dimensionality in dataset results in a rapid increase in the number of bitmaps. Further, the bitmap generation is quite an expensive step. Our work focuses on providing low latency by prefetching aggregated values in the cache.

There are several existing data storage systems that can support analytics. SciDB [28] is a column-oriented DBMS designed for multidimensional data management and analytics and provides support for complex analytics over multidimensional data. Systems such as Spark [35] and Shark [9] load datasets in memory in a distributed fashion and then allow further analytics. A common problem these systems face is lack of maintaining the geospatial relation between datapoints while storing data, which makes indexing slow.

Load-balancing and cache replacement problems have been explored in papers [16]–[18], [24], [25]. Paul et al. [25] describe a centralized control architecture for distributed coordinated caches to provide better web access times. Other work has proposed a load-balancing method that considers both localized access control and balanced load allocation [18]. This leverages a static caching approach that assigns hotspotted data to the server with higher processing power. The system also includes queuing theory and cache distribution strategy to achieve optimum processing time for data requests. However, it doesn't incorporate a cache replacement scheme to deal with changing hotspots. In [17], a replication strategy based on access characteristics of data is suggested to provide a high-speed caching. They propose a scheme to allocate cache by replicating hotspot tiles to multiple servers. Although, this can result in the fast depletion of the cache and may cause more delays due to frequent cache replacements.

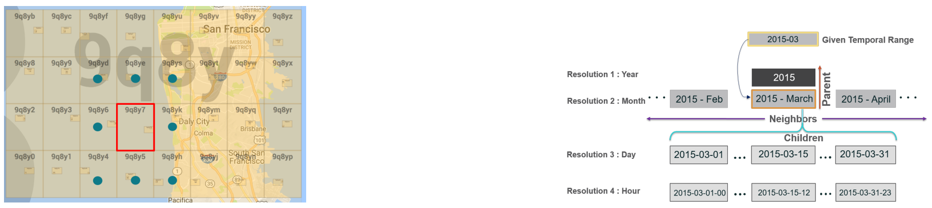
IV. Data Model and Query Evaluation [RQ-1]

STASH's data model is designed to efficiently store previously generated summary results in-memory from past queries and reuse them in case similar queries are performed by users in the future. This requires the summary data to be stored in the form of a collection of identifiable blocks or chunks with specific spatiotemporal bounds (we call them Cells) that can be rummaged and reused from the in-memory store.

STASH is logically organized as a multi-relational property graph with data that is aggregated at multiple levels of spatiotemporal resolutions. This graph is defined as $G_{STASH} =$

TABLE (I) Stash Cell Components

Content	Description	
Summary Data	Summary Statistics over datapoints lying in the spatiotemporal bounds of this Cell	
Spatiotemporal Bounds	The spatiotemporal bounds of current Cell	Geohash
		Time interval (eg. '2015-03')
Spatiotemporal Relationship Information (SRI)	Information identifying spatiotemporal parents, neighbors and children of current Cell	Spatial Parent(s), Temporal Parent(s) $\in E_H$
		Spatial Neighbors, Temporal Neighbors $\in E_L$
		Spatial Children, Temporal Children $\in E_H$



(a) Spatial (b) Temporal

Fig. (1) Spatiotemporal Relationship Among Individual Cells

$(V, \mathbb{E})$, where V is a set of vertices and $\mathbb{E} = \{E_H, E_L\}$ is a family of edge-sets. Vertices are labeled by their spatial and temporal information and have a set of properties represented as attributes. The set of hierarchical edges, E_H, represents an ordering between vertices that are one level apart in the spatiotemporal hierarchy. Hence, edge $e \in E_H$ with a source vertex V_S and a destination vertex V_D indicates that V_S's resolution is one spatial and/or temporal resolution greater than V_D. Therefore, V_S has a higher precision data aggregation than V_D and also the spatiotemporal bounds of V_D fully encloses that of V_S. Vertices with the same spatiotemporal resolution will be at the same depth/level in the hierarchy. The set of *lateral edges*, E_L, maintains the proximity of geospatial locations and temporal ranges between vertices in the same level. If there is an edge $e \in E_L$ between two vertices V_i and V_j, these vertices contain data for the two adjacent areas, i.e., their spatiotemporal bounds share boundaries. Each edge-set provides a distinctive traversal function and helps in the spatiotemporal neighborhood discovery for any region of Cells over G_{STASH}.

A. *Vertex: The* Stash *Cell*

Data aggregation involves grouping data points into bins of equal spatiotemporal extents and generating aggregated values for each attribute for all the points that fall within a certain bin. The array of aggregated attribute values for each bin is referred to as a **Cell**. A Cell is the minimum unit of data storage in Stash and represents a vertex of G_{STASH}.

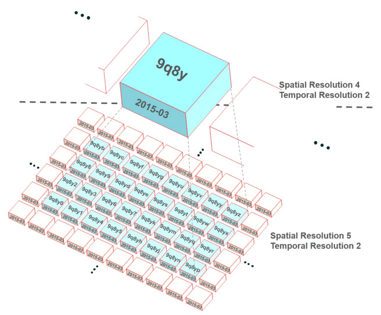
1) Properties of a Cell: Each Stash Cell, C_i, contains three main properties: (a) spatiotemporal labels, (b) aggregated summary statistics and (c) edge information($\{E_{H_i}, E_{L_i}\} | E_{H_i} \in E_H$ and $E_{L_i} \in E_L$), as shown in Table I. The *summary statistics* are the main content of a Cell and is the information returned to a client program in response to a spatiotemporal request. The spatiotemporal labels describe the scope including the spatial bounding box encoded as Geohash value and the chronological range for the observations. The edge information keeps the Cell aware of its immediate spatiotemporal neighborhood.

2) Nested Coverage: Stash *Cells* can be thought of as 3-dimensional cubes in the spatiotemporal space, with fixed bounds marked by their spatiotemporal labels. Cells connected by a hierarchical edge have nested spatiotemporal bounds, i.e., the bounds of the lower resolution Cell fully encloses the higher resolution Cell. Fig.2b shows a 2-dimensional representation of the nested bounds of Cells in the hierarchy. A Cell's spatiotemporal extent is inversely proportional to its spatiotemporal resolution. With every increase in spatial or temporal resolution, a single parent Cell gets broken into a fixed number of child Cells. For instance, Geohashes represent a hierarchy of successively higher-resolution spatial bounding boxes using a string of Base32 characters. So one spatial resolution increase splits each lower-resolution Cell into 32 equally-sized smaller Cells.

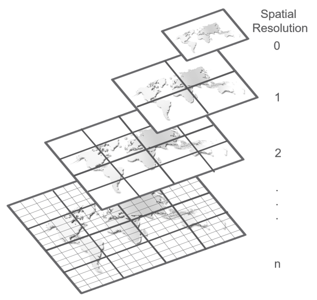
B. *Edge: The Inter-Cell Relationship*

Stash maintains two edge sets to represent distinctive relations between Cells. The hierarchical edges represent the spatiotemporal parent(s)/children of each Cell (see Table I). Each Cell can have 3 different parent precisions (one step lower spatial precision, one step lower temporal precision, and one step lower spatiotemporal precision). This refinement is applicable to the children nodes as well. The lateral edges help identify the spatiotemporal neighborhood of a Cell.

As shown in Fig.1 and Fig.2a, a Cell covering a geohash *9q8y7* and time *2015-03* has a spatial resolution of 5 (length of the Geohash) and temporal resolution 'Month'. The Cell has 8 adjacent spatial neighbors - *9q8yd, 9q8ye, 9q8ys, 9q8yk, 9q8yh, 9q8y5, 9q8y4, 9q8y6* (see Fig.1a) and 2 temporal neighbors *2015-02* and *2015-04*, which represent its *lateral edge*. Similarly parentage and children can be deduced from the Cells spatiotemporal information. For instance, the spatial parent of Geohash region *9q8y7* is 9q8y (spatial resolution



(a)



(b)

Fig. (2) Spatiotemporal Hierarchical Positioning of Cells

of 4) and each Geohash box encloses 32 nested Geohashes, which would represent the spatial children.

C. Hierarchical Cell Organization

To exploit the lineage and inter-relationship among the Cells, the STASH framework is organized in a hierarchical graph structure to support fast population and updates. The graph level for a given spatiotemporal resolution is calculated as $(n_j * n_t + n_i)$ where n_s and n_t are the total possible spatial and temporal resolutions, respectively and n_i and n_j are the current spatial and temporal resolution, respectively. Fig.2a gives a view of the relative positioning of two levels with varying resolutions.

D. Query Evaluation Strategy

The STASH graph is dispersed over the cluster; specifically, the main memory of the nodes. This dispersion attempts to maximize data locality, with the STASH graph at each node holding data related to that node. Although STASH is organized as a graph, the query evaluation does not rely on traditional graph traversal algorithms that often result in excessive network communications and iterations. Rather, STASH provides a set of composable vertex discovery schemes (through hierarchical and linear edge), instead of each Cell storing pointers to all its neighborhood Cells, that reduce the memory requirement and network communications significantly.

Each STASH level is dispersed over the zero-hop distributed hash table (DHT) based on their Geohash. Since the zero-hop DHT maintains the hosting information of the entire key ranges in each node, the query evaluation requires up to one query forwarding to locate the node holding the necessary segment. For a STASH cluster with N nodes, the computational cost to locate a Cell of a given level is $O(1)$.

Across multiple precision levels, STASH relies on **precision-level map (PLM)** to check for completeness of the in-memory data. The PLM is a memory-resident bitmap that associates the Cells contained in-memory for a given level to the actual data blocks in the distributed storage. In case of systems with real-time data, the PLM can be adjusted during an update to keep track of up-to-date Cells, so that stale data summaries are recomputed in case of future access. STASH consults the PLM to identify and retrieve missing chunks to complete the query evaluation.

V. LEVERAGING VISUAL NAVIGATIONAL PATTERNS [RQ-2]

A. Proximity-Aware Data Dispersion

Since our work focuses on rapid aggregation over large spatiotemporal data, preserving spatiotemporal proximity and inter-Cell relationships in our data structure provides data locality. As mentioned in Section I-B, spatiotemporal proximity is preserved by DHT-based distribution of the STASH graph.

To preserve Cell relationships, we leverage the strong spatial and temporal locality of access [11] which is characteristic of spatiotemporal user requests. **Spatial locality** implies that if a spatiotemporal region is accessed, its neighborhood also has high chance of future access, while **temporal locality** implies that a region's popularity is directly proportional to its probability of being accessed in the near future, which is in line with *Zipf's Law* [17].

B. Collective Caching

STASH provides a query optimization scheme tailored for explorative analytics through OLAP operations [8] for multi-dimensional data. It adapts popular data mining operations that are encapsulated within OLAP operations, particularly those related to visualization, such as slicing, dicing, panning, zooming, drilling-down, and rolling-up. *Slicing* is the act of picking a subset by choosing a single dimension. *Dicing* produces a subset by allowing the analyst to constrain inclusions based on specific values across multiple dimensions. *Panning* allows users to explore a neighborhood. *Drill-down* and *roll-up* allow users to navigate through more finer-grained resolution level and coarser-grained resolution, respectively.

These operators are translated to a query with expressive predicates specifying spatiotemporal coverage along with the data aggregation requirements. A sequence of operators often involves partially overlapping or nested queries (e.g., a series of panning operators may involve overlapping areas and zooming or rolling-up results in nested queries). Although existing scalable databases support sophisticated aggregation schemes with caching mechanisms [1], the query output is not **reusable** by other partially similar queries or by other users. STASH's in-memory cache is collectively built through query evaluations from multiple users. Any subsequent query will be evaluated over the cached values first. Disk access is required only if, (a) there are missing values for completing query evaluation, and (b) those missing values are not available by computing from the existing cached values.

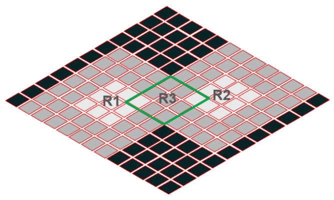


Fig. (3) STASH Freshness Dispersion Scheme

C. Cell Replacement Strategy

The total number of possible Cells is much larger than the number of Cells that can be persisted in-memory. The threshold for the total number of Cells allowed in STASH is configurable and limited. For our Cell replacement strategy, rather than focus on the demand of Cells individually by only calculating their frequency of access, we focus more on the spatiotemporal **regions of interest**, i.e. the regions that are experiencing the most user queries at a particular instant.

1) Cell Freshness: The effectiveness of STASH lies in its ability to maintain the most relevant regions in the memory, and to efficiently detect stale Cells and swap them out for more requested regions, in case we reach a threshold due to overpopulation of Cells.

In order to determine which Cells to persist in-memory in case of a threshold breach, we use the metric of **freshness** to evaluate the importance of a Cell in the STASH Graph. *Freshness* is calculated as the product of the number of accesses to a Cell (updated every time it gets accessed), and a time decay function. Hence, both frequency and recency of access are contributors to the *freshness* of a Cell. Cells in STASH are replaced based on this freshness score.

2) Cell Replacement Based on User Access Patterns: In accordance with the spatial and temporal locality of access patterns, when a request for a spatiotemporal region comes in, we mark both the set of Cells in that region and the immediate spatiotemporal neighborhood of that region as being of future interest, to prepare for possible overlapping requests in the immediate future and update their *freshness*.

To explain freshness dispersion among Cells, we refer to Fig.3, that gives a two-dimensional view of a particular spatiotemporal resolution and the Cells contained therein. Let us say that spatiotemporal regions R_1 and R_2, highlighted by light-colored Cells, have been accessed by one or more end-users recently. Temporal locality of access dictates that Cells accessed recently have a higher probability of access in the near future. Therefore, in our freshness dispersion scheme, when regions R_1 and R_2 get accessed, we increase their freshness (by, say, f_{inc}, which is configurable). Also, spatial locality of access suggests that if regions R_1 and R_2 are currently of interest, their spatiotemporal neighborhood will also be a region of interest. To reflect this property, we also disperse a fraction of f_{inc} to the Cells in the immediate neighborhood of R_1 and R_2 (grey Cells). This scheme prevents the spatiotemporal neighborhood from being deemed stale, even though they might not have been accessed recently.

Practically speaking, the reason behind focusing on regions instead of the queries' exact spatiotemporal bounds is two-fold. First, from a single user's perspective, an incoming query is one among a series of queries to the server, each corresponding to an action over the front-end UI. Rather than focus on the exact region of query, our methodology determines the combined spatiotemporal neighborhood of interest (both light and gray areas in Fig.3), since those are the regions likely to be requested in subsequent requests. Second, considering multiple users and their queries, spatial locality of access dictates that their requests will be focused around small clusters of the total spatiotemporal space. Thus, focusing on the spatiotemporal neighborhood of a single query's extent as well helps us prepare for similar queries in the immediate neighborhood.

STASH Cell replacement involves evicting the Cells with the lowest freshness score till the capacity goes below a safe limit. The freshness dispersion scheme described above helps entire regions that are heavily accessed to be persisted in memory during replacement, instead of disconnected patches that would reflect the actual query areas that were fetched but might hamper the performance and latency of future queries.

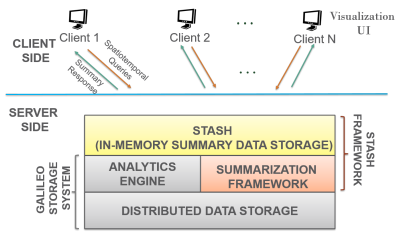


Fig. (4) STASH System Architecture

3) Advantage of the Hierarchical Graph Organization: Updating the freshness of a set of Cells belonging to a user-requested region out of a large collection of in-memory Cells and their spatiotemporal neighborhood is a time-consuming operation that can potentially slow down the query evaluation. To accomplish the above goal efficiently, we need to be able to (1) effectively isolate and quantify regions that are in demand with respect to regions that are less requested and (2) quickly identify the spatiotemporal neighborhood of a requested region. The hierarchical organization of STASH Cells allows us to perform the aforementioned functionalities fast and effectively. First, it allows us to isolate the set of Cells belonging to the current spatiotemporal resolution, as well as their parent and child-level resolutions, thus narrowing down the scope of relevant Cells. Second, once Cells have been identified as part of a queried region, we can easily use their lateral and hierarchical relationship(edges) to find the Cells in their immediate spatiotemporal neighborhood and update their freshness values in a fast manner.

VI. System Overview

Stash is designed as an in-memory data storage framework that is positioned as an intermediary between the users interacting with a lightweight front-end UI and a back-end distributed storage and analytics engine. The front-end UI internally converts each user interaction into a request to the back-end server and parses and displays the server's responses on the display window. Fig.4 demonstrates the various components of our system.

A. Front-end Visualization UI

In our system, the front-end visualization is performed by using Grafana [2]; we can interoperate with any visualization framework that is capable of parsing and displaying summarization responses in JSON. Grafana is an open-source visualization and metric-analysis tool and we have used its WorldMap panel to display spatiotemporal results.

B. In-Memory Summary Stash-ing Framework

Stash acts as a caching middleware between incoming user-requests and the back-end distributed analytics engine.

C. Back-end Distributed Storage

In our implementation, for the back-end storage and analytics engine, we use Galileo [21], a distributed storage and analytics framework for large multidimensional, spatiotemporal datasets. Galileo is a zero-hop Distributed Hash Table(DHT) based storage system that uses Geohash [22] to generate data partitions that store and colocate geospatially proximate data points. The granularity of the coverage of a data block is determined by the length of geohash code managed by the nodes. Stash builds on Galileo's distributed query evaluation capability to efficiently query and then summarize over data points that match a user query at varying spatiotemporal resolutions.

VII. Autoscaling for High Throughput Query Evaluation [RQ-3]

The spatial and temporal locality of user access patterns makes it highly likely for **hotspots** to emerge over the distributed data storage system [33]. A large number of queries focused over a small spatiotemporal portion of the entire data space would lead to only a few nodes servicing a large chunk of queries, leading to a bottleneck. Also, these hotspots are dynamic [15]. In Stash, individual nodes adapt to potential hotspots in a dynamic and decentralized fashion.

A. Dynamic Clique Replication

Stash's hotspot handling focuses on a dynamic replication scheme for the most accessed spatiotemporal **regions**. Whenever the workload (determined in our case through the pending requests queue size) on a node in the cluster crosses a configurable threshold, that node initiates a **Clique Handoff** for the most active spatiotemporal regions on it. Hotspotted regions in Stash are demarcated in terms of **Cliques**(explained in Section VII-B), which represent the most active set of Cells in Stash and acts as the unit of data replication and transfer in Stash.

Clique Handoff is the decentralized process of the **hotspotted node** finding the most suitable candidate node (called the **helper node**) to house replicas of its *hottest* Cliques, so that future queries have a high chance of accessing fully replicated regions, thus alleviating some of the processing load on the hotspotted node. A helper node maintains two Stash graphs - one local and one guest (containing replicated Cells from other hotspotted node(s)). In case of updates to the actual data storage, the PLM helps identify the stale replicas, which are discarded from serving redirected requests.

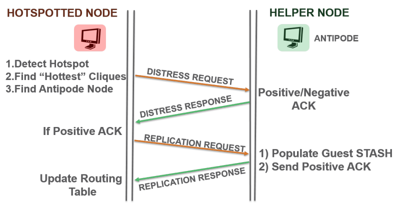


Fig. (5) Hotspotted Region Handoff

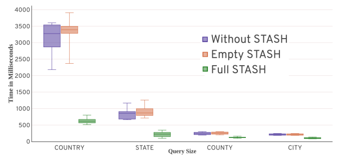
B. Clique Handoff Process

The Clique Handoff process involves the following steps, as depicted in Fig.5:

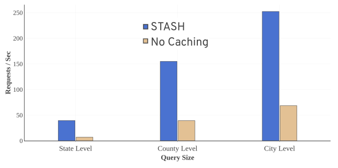
1) Hotspot Detection: In our implementation, a node deems itself to be hotspotted when the number of pending requests in its message queue crosses a configured threshold.

2) Top Cliques Calculation: The hotspotted node attempts to find the spatiotemporal regions that are causing majority of its workload in the form of Cliques in its Stash graph with the highest cumulative *freshness*. We define Cliques, here, as a subgraph of Cells from the Stash graph of a pre-configured size (depth). For example a Clique of depth 2 would consist of a Cell C_i and all its children Cells and their children Cells to calculate their cumulative freshness. Cliques are identified by the spatiotemporal label of their topmost parent Cell.

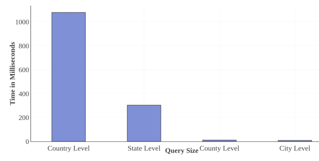
We set the maximum number of replicable Cells at a time to a preset amount, say N. The hotspotted node searches its Stash graph to find the top K Cliques whose cumulative size is $\leq N$. The hierarchical structure of Stash graph makes it efficient to identify the Cells that would be in a given Clique. These top K Cliques are subject to replication from a hotspotted to helper node(s).



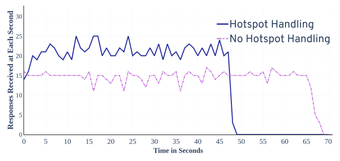
(a) Latency vs Query Size



(b) Throughput Improvement



(c) Maintenance Overhead



(d) Autoscaling

Fig. (6) Performance Evaluation of STASH: (a) and (b) show effects of query size on its latency and throughput, respectively; (c) compares STASH maintenance time for different query sizes and (d) shows the improvement in throughput for STASH's replication mechanism over normal execution during hotspot.

3) Antipode Node Selection: The candidate for a helper node is calculated separately for each Clique. Since hotspots tend to be concentrated in small pockets in the spatiotemporal space, our goal is for Clique replicas to be housed on nodes whose domain is the most isolated from the current hotspotted region. In our implementation, we look for a spatiotemporal region that is diametrically on the other side of the total spatial scope of the storage cluster. The idea is similar to the concept of *antipode* of a coordinate which is another coordinate on the diametrically opposite side of the globe. We call the node handling this diametrically opposite region the **antipode node**.

Using a Clique's geohash, we find its geohash antipode and then use the DHT's partitioner to identify the antipode node and send it a *Distress Request*. If it is not itself hotspotted and its guest tree can accommodate the incoming Cells, the antipode node sends a *positive acknowledgement*. In case of a negative acknowledgement, the hotspotted node repeats the above process for another geohash region in a random direction around the antipode geohash.

4) Replication Request/Response: On a positive acknowledgment, the hotspotted node sends a *Replication Request* to the helper containing the Clique(s) to replicate. The helper inserts the Clique into its guest STASH graph and replies with a successful Replication Response.

5) Routing Table Population: The hotspotted node maintains a routing table of Cliques that are replicated at helper nodes, along with a bitmap of the actual Cells contained in the Clique. This routing table is populated upon receiving a successful Replication Response from a helper node.

C. Query Evaluation under Hotspot

In a hotspot situation, a user query is first checked against entries in the routing table and if the spatiotemporal region of the user query is found to be fully replicated at another helper node, the user request is probabilistically rerouted to the helper node, thus reducing the load on the hotspotted node. At the helper node, the relevant Cells are fetched from its guest STASH graph, just as it would be from a local STASH graph.

D. Replication and Cleaning

Each node has a pre-configured cooldown time after hotspot handling. If the hotspot persists after this cooldown time, the Clique Handoff process is repeated and another set of replicas of active Cliques are created on candidate helper nodes.

The guest STASH graph entries also get purged if they are not requested to be persisted within a configurable amount of time. Stale routing-table entries also get purged from the hotspotted node after a pre-configured period signifying the retreat of hotspot.

VIII. Empirical Benchmarks and Discussion

A. Experimental Setup

To evaluate compute-intensive operations with high-density observations, we profiled our system while performing OLAP operations with spatiotemporal data on a cluster of 120 nodes. Each node in our distributed cluster is an HP Z420 with the following configuration: 8-core Xeon E5-2560V2, 16 GB RAM and 1 TB disk. The data is partitioned uniformly over the cluster based on the first 2 characters of their Geohash.

To contrast performance with other geospatial caching systems, we have used Elasticsearch [1] on a cluster with 3 master nodes and 120 data nodes. To achieve horizontal scalability and parallelization, the index was split into 600 shards. Three types of caches that were maintained stored the query results, aggregations, and field values on a node.

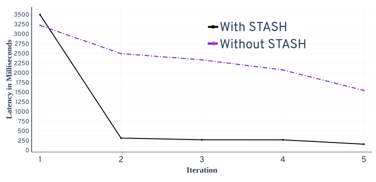
Throughout our experiments, we refer to 4 groups of spatiotemporal queries as country, state, county or city level. These represent 4 query sizes that vary in their spatial extent (Query_Polygon) but have a fixed temporal extent which is *2015-02-02* (Query_Time). The spatial extent of the 4 query groups is set using a random rectangle over the data's entire spatial coverage with latitudinal and longitudinal extent of (16°,32°), (4°,8°), (0.6°,1.2°) and (0.2°,0.5°), respectively. The requested spatial and temporal resolutions are 6 and 'Day of the Month', respectively, unless otherwise specified.

B. Dataset

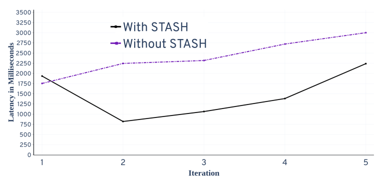
The dataset used for our experiments is sourced from the NOAA North American Mesoscale (NAM) Forecast System [3]. The NAM dataset (∼1.1 TB unprocessed) contains atmospheric data collected several times per day for 2013, globally including features like surface temperature, relative humidity, snow and precipitation.

C. Distributed Query Evaluation with STASH

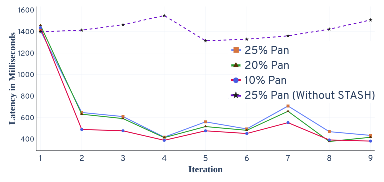
1) Query Evaluation Time ***(RQ-1)****:* We profiled latency improvement with the STASH framework by tracking average latencies for queries of varying sizes for 3 scenarios – the



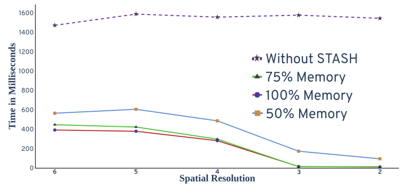
(a) Iterative Dicing - Descending



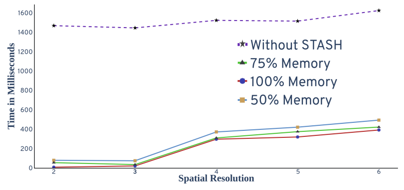
(b) Iterative Dicing - Ascending



(c) Panning



(d) Drill-Down



(e) Roll-Up

Fig. (7) Query Performance Evaulation for Several Visual Analytics Operations

simple Galileo storage system, empty STASH graph with no Cells (worst-case), and STASH graph with all necessary Cells in-memory (best-case - duplicate query). This kind of querying for a subset of the total spatiotemporal extent of the data reflects the *dicing* operation in our system.

Fig.6a shows that STASH with all necessary Cells in-memory outperforms the other two scenarios with ∼**5x improvement** over no STASH scenarios for large query sizes such as country and state. A fully populated STASH graph demonstrably transforms even large queries to interactive operations.

2) Overhead of STASH *Maintenance:* In Fig.6a, the average latency in the worst case scenario is slightly more than in the no STASH case, which can be explained by the overhead in the unsuccessful look-up for matching Cells in the graph and then attempting retrievals from disk. The population of Cells fetched from disk to memory is done at the back-end in a separate thread. Fig.6c depicts the cold-start scenario where all the Cells from a query have to be inserted in-memory and the time taken population that goes down considerably with query size since lesser Cells are to be inserted in STASH.

D. *Query Optimization for Visual Exploration* ***(RQ-2)***

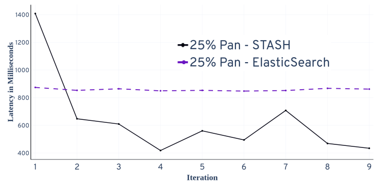
In practical scenarios, fragments of the query's spatiotemporal extent will be contained in STASH graph while the remainder needs to be fetched from disk as follows:

1) Iterative Dicing: To simulate the user action of sequentially increasing and decreasing the query area, we have implemented ascending and descending iterative dicing, as shown in Fig.7b and Fig.7a respectively. It shows a sequence of 5 queries that, keeping the spatiotemporal resolution fixed, vary the Query_Polygon size in either ascending order or descending order. We can see that descending iterative dicing performs much better for a STASH-enabled system since a larger area (country level) is fetched in the first query and then, iteratively, a subset of the first query (20% spatial area reduction) gets queried (final query having size ∼(5.2°,10.4°)) - leading to all necessary Cells existing in memory from the second query onwards. The ascending version is the previous set of queries executed in reverse order. Here, as the spatial extent increases, a fraction of the relevant Cells are found in-memory, which does lead to improved performance over the basic system, but not to the extent of the descending version.

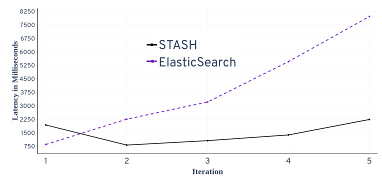
2) Zooming: We replicate the scenario of a user sequentially increasing or decreasing the resolution of a view area by two sets of experiments - drill-down (zoom-in), where a user starts with a lower spatial resolution of 2 of a state-level area and then recursively increases the resolution to 6 that incurs ∼32 fold increase in the number of possible Cells at each step. Roll-up (zoom-out) is the reverse of the drill-down operation. To compare the performance of our system in scenarios with varying amount of relevant Cells in-memory, we have randomly stacked the STASH graph with regions covering 50%, 75% and 100% of all the relevant Cells.

Fig.7d and Fig.7e contrasts the latency of the drill-down and roll-up scenarios, respectively, for a STASH enabled system against the basic system. As expected, more the amount of relevant Cells in-memory, the better the latency. However, in all scenarios with partial information, we see at least **40% improvement in latency** over a system without STASH.

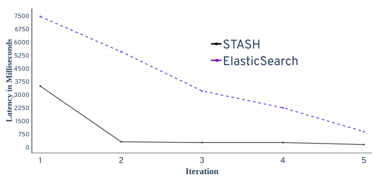
3) Panning: We replicate panning in our experiments by starting with a state-level query and moving the rectangle by a certain amount (10%, 20%, 25%) in 8 possible directions around the starting rectangle. So, the first query encounters an empty STASH graph and then, from the second query onwards, a fraction of the necessary Cells should exist in-memory. The results in Fig.7c support our assertion. We see that the basic analytics system has uniformly high latency, whereas, that in STASH enabled system is considerably low. The lower the amount of pan, the larger is the overlapping area between two consecutive queries, which would benefit a STASH enabled system, as validated by Fig.7c. Also, the comparison of 25% pan scenario between a basic and a STASH enabled system shows considerable improvement ranging from **73%-60% reduction in latency**.



(a) Panning



(b) Iterative Dicing - Ascending



(c) Iterative Dicing - Descending

Fig. (8) Contrasting STASH's Performance with ElasticSearch for Common Visual Analytics Operations

4) Throughput: Fig.6b shows the throughput of a STASH-enabled system vs that of a basic system. This experiment involves firing 10,000 county-level requests over the cluster which are created by selecting 100 random rectangles (of sizes state, county and city) over the globe and then randomly panning around each by 10% in any random direction 100 times, to replicate spatiotemporal locality of requests. The throughput is calculated based on the total time taken for the last request to be executed successfully. A STASH-enabled system shows **5.7x, 4x and 3.7x improvement in throughput** for state, county and city-level queries, respectively.

*E. Autoscaling **(RQ-3)***

Fig.6d contrasts the performance of STASH's replication scheme against STASH without dynamic replication under skewed traffic. We simultaneously executed 1000 county-level requests, by randomly panning around a random starting point, to emulate the hotspot scenario of sudden interest over a single region from multiple users. Our system was configured to initiate Clique handoff with pending requests of over 100. To compare improvement caused by a replication operation, the cooldown time was set high. Fig.6d shows the number of responses received each second from the start. We can see that STASH with a dynamic replication scheme processes larger number of queries per second and finishes all tasks $\sim$ 20 seconds before STASH without dynamic replication.

F. Comparison with ElasticSearch

We contrasted STASH's performance against ElasticSearch, which has its own caching system, with some of the previously OLAP scenarios with consecutive overlapping requests.

1) Panning: The panning scenario when replicated on ElasticSearch gives results as shown in Fig.8a. We can see that STASH shows better improvement in performance, whereas ElasticSearch's latency improves slightly. At each step with the latency-reduction with respect to the latency of the first request with STASH ranges between $\sim$70% and 49.7%, whereas that of ElasticSearch stays between $\sim$2% and 0.6%. Also, the second query onwards, STASH's latency is significantly lower which demonstrates better management of in-memory data in case of overlapping queries.

2) Dicing: Fig.8b and Fig.8c compares the results of the ascending and descending iterative dicing experiments, as mentioned above, between STASH and ElasticSearch. Here also, we see that STASH **achieves a much steeper drop in latency from the second query onwards** by efficiently utilizing the common Cells stored in-memory for the subsequent queries.

IX. CONCLUSION AND FUTURE WORK

We described our framework, STASH, for harnessing spatiotemporal query evaluations and their underlying patterns to assist timely, high-throughput evaluation of future queries in support of effective visual explorations of spatiotemporal data. Our methodology is agnostic to the underlying storage framework, and STASH can be configured on top of a distributed storage system to help facilitate visualization and preserve interactive responsiveness.

RQ-1: The hierarchical structure of STASH graph with Cells grouped by their resolutions while encoding lateral and hierarchical edge relationships facilitates fast identification of relevant Cells for a query. The PLM can be effectively leveraged to calculate the completeness of in-memory data in relation to the query, which helps reduce disk I/O by precisely identifying the particular portions that are missing from STASH.

RQ-2: STASH's Cell replacement scheme dynamically adapts to the spatial and temporal locality of user-accesses by dispersing freshness to the spatiotemporal neighborhood based on the query scopes. This allows the framework to house the most relevant spatiotemporal regions in-memory, substantially reducing the number of disk accesses and the amount of data to be processed from disk during query evaluations. This, in turn, reduces query latencies and improves throughput.

RQ-3: Our dynamic replication scheme effectively handles skews in access patterns. By ensuring that heavily accessed spatiotemporal neighborhoods are replicated, and replicating them on nodes farthest from the hotspotted region, we ensure effective utilization of resources within a cluster.

A. Future Work

Our proposed future work builds on the work describe here. First, a smaller-capacity STASH graph at the front-end can greatly reduce latency in case users tend to browse a narrow spatiotemporal region, thus reducing the number of queries needed to be evaluated at the back-end. Second, constructing a trained model that accurately predicts a user's access pattern can assist in the construction of prefetching queries that augment regions that the model predicts would be of interest in future with the region to be requested currently. More importantly, this can help reduce the number of interactions the front-end needs to have with the server.

References

[1] *Elastic Search*, 2019. https://www.elastic.co/guide/en/elasticsearch/reference/6.2/index.html.

[2] *Grafana Labs*, 2019. https://grafana.com/grafana.

[3] *National Oceanic and Atmospheric Administration, The North American Mesoscale Forecast System*, 2019. http://www.emc.ncep.noaa.gov/index.php?branch=NAM.

[4] *Redis for Geospatial Data*, 2019. http://lp.redislabs.com/rs/915-NFD-128/images/WP-RedisLabs-Geospatial-Redis.pdf.

[5] *SAP Lumira – Introduction For Beginners*, 2019. https://blogs.sap.com/2014/08/08/sap-lumira-introduction-for-beginners/.

[6] *Tableau Desktop*, 2019. https://www.tableau.com/products/desktop.

[7] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1363–1375. ACM, 2016.

[8] A. Berson and S. J. Smith. *Data warehousing, data mining, and OLAP*. McGraw-Hill, Inc., 1997.

[9] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: fast data analysis using coarse-grained distributed memory. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 689–692. ACM, 2012.

[10] N. Ferreira, J. Poco, H. T. Vo, J. Freire, and C. T. Silva. Visual exploration of big spatio-temporal urban data: A study of new york city taxi trips. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2149–2158, 2013.

[11] D. Fisher. Hotmap: Looking at geographic attention. *IEEE transactions on visualization and computer graphics*, 13(6):1184–1191, 2007.

[12] M. García and B. Harmsen. *Qlikview 11 for developers*. Packt Publishing Ltd, 2012.

[13] P. Hanrahan. Analytic database technologies for a new kind of user: the data enthusiast. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 577–578. ACM, 2012.

[14] J. Heer and B. Shneiderman. Interactive dynamics for visual analysis. *Queue*, 10(2):30, 2012.

[15] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 34. ACM, 2008.

[16] R. Li, J. Fan, X. Wang, Z. Zhou, and H. Wu. Distributed cache replacement method for geospatial data using spatiotemporal locality-based sequence. *Geo-spatial Information Science*, 18(4):171–182, 2015.

[17] R. Li, W. Feng, H. Wu, and Q. Huang. A replication strategy for a distributed high-speed caching system based on spatiotemporal access patterns of geospatial data. *Computers, Environment and Urban Systems*, 61:163–171, 2017.

[18] R. Li, Y. Zhang, Z. Xu, and H. Wu. A load-balancing method for network giss in a heterogeneous cluster-based system using access density. *Future Generation Computer Systems*, 29(2):528–535, 2013.

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

[20] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

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

[22] G. Niemeyer. *Geohash*, 1999. http://www.geohash.org/.

[23] C. A. Pahins, S. A. Stephens, C. Scheidegger, and J. L. Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE transactions on visualization and computer graphics*, 23(1):671–680, 2017.

[24] S. Pan, L. Xiong, Z. Xu, Y. Chong, and Q. Meng. A dynamic replication management strategy in distributed gis. *Computers & geosciences*, 112:1–8, 2018.

[25] S. Paul and Z. Fei. Distributed caching with centralized control. *Computer Communications*, 24(2):256–268, 2001.

[26] L. Santos, J. Coutinho-Rodrigues, and C. H. Antunes. A web spatial decision support system for vehicle routing using google maps. *Decision Support Systems*, 51(1):1–9, 2011.

[27] V. Sharma. Getting started with kibana. In *Beginning Elastic Stack*, pages 29–44. Springer, 2016.

[28] M. Stonebraker, P. Brown, D. Zhang, and J. Becla. Scidb: A database management system for applications with complex analytics. *Computing in Science & Engineering*, 15(3):54, 2013.

[29] Y. Su, Y. Wang, and G. Agrawal. In-situ bitmaps generation and efficient data analysis based on bitmaps. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 61–72. ACM, 2015.

[30] W. Tao, X. Liu, Ç. Demiralp, R. Chang, and M. Stonebraker. Kyrix: Interactive visual data exploration at scale. CIDR, 2019.

[31] J. J. Van Wijk. The value of visualization. In *VIS 05. IEEE Visualization, 2005.*, pages 79–86. IEEE, 2005.

[32] R. Wesley, M. Eldridge, and P. T. Terlecki. An analytic data engine for visualization in tableau. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1185–1194. ACM, 2011.

[33] C. Yang, M. Goodchild, Q. Huang, D. Nebert, R. Raskin, Y. Xu, M. Bambacus, and D. Fay. Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing? *International Journal of Digital Earth*, 4(4):305–329, 2011.

[34] B. Yost, Y. Haciahmetoglu, C. North, and C. North. Beyond visual acuity: the perceptual scalability of information visualizations for large displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 101–110. ACM, 2007.

[35] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.