Periscope: A Framework for Visualizations of Multiresolution Spatiotemporal Data at Scale

Everett Lewark[1], Matthew Young[2], Paahuni Khandelwal[3], Sangmi Lee Pallickara[4], Shrideep Pallickara[5]
Department of Computer Science, Colorado State University, Fort Collins, CO
Email: elewark@colostate.edu[1], asterix@rams.colostate.edu[2], paahuni@colostate.edu[3],
Sangmi.Pallickara@colostate.edu[4], Shrideep.Pallickara@colostate.edu[5]

2024 IEEE International Conference on Big Data (BigData) | 979-8-3503-6248-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/BigData62323.2024.10825317

The crux of this study is to support browser-based visualizations of spatiotemporally evolving phenomena. Such phenomena arise in myriad domains spanning terrestrial, oceanic, and atmospheric processes. The data are voluminous, have diverse representational formats and projection systems, and are multivariate. We rely on a novel mix of tiling, caching, compression, perceptual limits, speculative prefetching, and dynamic generation of tiles. Our refinements at the client and server-side work in concert with each other to leverage client-side resources, minimize duplicate processing, and effective prefetching to ensure interactive explorations at scale. Our benchmarks profiled several aspects of our methodology and demonstrate the suitability of our refinements.

***Index Terms*—spatiotemporally evolving phenomena, visualizations, tiling, caching, speculative prefetching, streaming**

I. Introduction

Data volumes have grown in several domains. This growth has been fueled by the proliferation of sensing simulations and data harvesting via logging mechanisms. As data volumes have grown, they offer opportunities to extract insights and patterns from them to inform decision making. Approaches to doing so include launching ad hoc tasks, fitting models to the data using AI/ML based approaches, and visualization. This study explores issues and methodological aspects to support visualizations over voluminous datasets.

The class of datasets we consider are spatiotemporal datasets. Data items in such datasets include geocodes identifying the location, or spatial extent, for which the data hold. Data items, which can be multivariate, also include a timestamp identifying when the observations were made. Such data occur frequently in the modeling and sensing of naturally occurring phenomena in domains such as geosciences, agricultural systems, atmospheric sciences, meteorology, ecology and environmental sciences.

Users leverage visualizations to support several types of explorations. This includes panning and zooming in and out spatially, alongside temporal drill-downs and roll-ups. They may also be interested in visualizations at different granularities that combine exploring spatial variation of phenomena at coarser scales while at the same time probing subtle regional variations based on finer-scale visualizations. Finally, users may also be interested in contrasting spatial variation across different timesteps or overlaying different phenomena.

Browser-based rendering supports simplicity and democratization of accesses without the need for specialized hardware or software requirements. For this reason, we investigate techniques to provide interactive spatiotemporal visualizations that users can access through a web browser.

A. Challenges

There are several challenges in effective visualizations of spatiotemporally evolving phenomena. These include:

1) Voluminous datasets: The datasets we consider are voluminous. As data volumes increase, this may place computational, memory, and data transmission overheads on both the client and server(s).
2) Data at different resolutions: Users may wish to layer phenomena. The phenomena may be sensed or reported at different resolutions, that is, the same geographical extent may be rendered at different resolutions.
3) Projection systems of the data may vary and need to be reconciled. Spatial datasets are encoded in a plethora of encoding formats and spatial projection systems. Failure to reconcile projection systems introduces distortions that compromise the fidelity of the rendering operations.
4) Spatiotemporal alignment of data during visualizations. Because data items have associated spatial and temporal dimensions, the data items need to be collated and aligned prior to rendering.
5) Interactivity: The effectiveness of visualization explorations is predicated on interactivity despite the data volumes and data resolutions involved.

B. Research Questions

Within the broader context of visualizing spatiotemporally evolving phenomena, we explore these research questions:

RQ-1: How can we preserve interactivity during explorations? This interactivity must be preserved despite the volumes, geographical extent, and data resolutions.

RQ-2: How can we cope with data volumes and ensure that the backend scales with increased data volumes? The server-side must cope with increased data volumes and clients rendering phenomena. Because the visualization processes are

This research was supported by the National Science Foundation (1931363, 2312319), the National Institute of Food Agriculture (COL014021223), and an NSF/NIFA Artificial Intelligence Institutes AI-CLIMATE Award [2023-03616].

colocated with other processes and tasks that are executing server side, inefficiencies impact such colocated processes.

RQ-3: How can we effectively contrast phenomena at different spatial resolutions and timescales? Users are interested in contrastive abilities that must be intuitive and interactive.

C. Approach Summary

Our methodology includes a carefully calibrated mix of (1) data staging, (2) tiling and dynamically generating tiles, (3) managing perceptual limits, (4) leveraging caching on the client and server side, and (5) speculative prefetching schemes informed by client exploration trajectories. We explore these ideas in the context of our visualization engine, PERISCOPE, that we have designed from the ground-up to support explorations of our spatiotemporally evolving phenomena.

We stage our datasets to manage the competing pulls of dispersion and colocation. Our staging is based on deterministic spatial partitioning. In this study, we use quadtiles, but we can leverage other deterministic spatial partitioning schemes such as geohashes. Our partitioning and dispersion scheme ensures that data from a spatial extent are colocated on the same machine, but data from different spatial extents are stored on different machines to ensure load balancing.

Regardless of the zoom level, the viewport is rendered as a collection of tiles. Breaking the viewport into tiles allows our framework to preferentially cache, evict, retrieve, and stream tiles comprising the viewport. This makes effective use of the cache at the client and server side while ensuring responsiveness. We precompute some coarse-grained tiles, but intermediate representations at different zoom levels are computed dynamically. The viewport for the coarsest resolution is precomputed and cached; this is incrementally refined by either computing intermediate resolution tiles or fetching the highest resolution tiles. We choose WebP and GeoTIFF as the image formats for representing tile data. WebP has two key advantages: the ability to manage tradeoffs associated with image quality and compression, and its support for transparency, which is important during layering operations. The spatial projection for the tile is reconciled to the Web Mercator projection system used by our visualization engine.

Both the client and server side maintain caches. Our caches—both client-side and distributed server-side—are populated and evicted with tiles based on the exploration patterns. We rely on computing exploration trajectories during panning, drill-downs, and roll-ups. During the shorter dwell times as a user orients themselves with the newly rendered data, prefetching operations are performed away from the critical path. The prefetched tiles are placed in the cache and subsequent accesses to those tiles do not incur disk IO or network transmission costs.

Together, caching, prefetching and speculative paths, tiling, and streaming based incremental refinements allow us to support interactive explorations of spatiotemporally evolving phenomena. We validate our ideas in the context of a soil moisture content (SMC) dataset and its accompanying spatiotemporal variation. SMC, which is the volume of water per unit volume of soil, is one of the most important parameters in agriculture. SMC in the crop's root-zone is directly related to the water available for root-water uptake and transpiration and contributes to crop health and yields. The SMC dataset is available for a wide swath of the southwestern U.S.. We also provide contrastive capabilities with related, ancillary phenomena such as the National Land Cover Database (30m) [1], Copernicus Digital Elevation Maps (30 m) [2], SMAP satellite data (36 km resolution) [3], and SMC measurements from previous days.

D. Paper Contributions

We describe our solution to enabling visualizations of spatiotemporally evolving phenomena. Our contributions include:

1) Client and server-side caches that work in concert with each other to minimize disk I/O and network transmission costs. The caches also include mechanisms to preferentially cache key data while relying on usage patterns for evictions.
2) A speculative prefetching scheme that exploits user dwell times to retrieve tiles that a user is likely to use next. As our results demonstrate, this improves interactivity during visualizations.
3) An incremental refinement scheme that leverages streaming of finer resolution tiles to incrementally update a viewport comprising coarse grained tiles.
4) Leveraging human perceptual limits to dynamically generate or retrieve tiles based on user explorations while minimizing duplicate work performed.

E. Paper Organization

The remainder of this paper is organized as follows. Related work is discussed in section 2. In section 3 we describe our methodology with performance benchmarks in section 4. Finally, in section 5 we outline conclusions and future work.

II. RELATED WORK

Large scale visual analytics is an integral approach to decision-making, combining visualization, human factors and data analysis. As the amount of data grows explosively, from observational instruments to social media posts, several efforts in both academia and industry have focused on visual analytics. This includes traditional visual analytics products such as Tableau [4], Spotfire [5], JMP(SAS) [6], and NetCHART [7]. There are also a number of toolkits such as InfoVis [8], Prefuse [9], and Improvise [10]. The functionalities popularly supported by these systems include statistical summaries (e.g. sum, average, or counts), data handling (e.g. join or group-by), charts (bar, line, pie chart, or histogram), heat maps, parallel coordinates, and scatterplot matrices. One of the recent trends is providing advanced analysis methods such as data modeling (e.g., clustering, classification, network modeling, and predictive analysis), and data projection (e.g., Principal Component Analysis, Multidimensional Scaling and Self Organizing Maps). There are existing products particularly

targeting voluminous datasets, such as Nanocubes [11] or imMens [12] that aim for visualization of voluminous datasets, and allow for real-time queries over the datasets as well. Other approaches, such as Hashedcubes [13], target reduction of memory consumption profiles as the primary precursor to real-time visualization. The key objective in PERISCOPE is rendering spatiotemporally evolving phenomena at scale alongside spatial and temporal constructs that facilitate them.

There has also been a substantial amount of research into tiled map rendering approaches, including benchmark techniques, data stores, and caching. In [14], Liu and Nie compare the performance of a Web Map Service (WMS) server with and without a tiled GeoWebCache server in front. While a tiled map was initially slower to render than WMS, which normally renders the entire screen at once, successive requests populated the cache and caused performance to speed up dramatically [14]. Wu et al. also apply GeoWebCache to decrease load time, but use an Nginx reverse proxy for load balancing and a NoSQL MongoDB instance for storing tile images [15].

A prior analysis by Fisher [16] visualized tile usage patterns using a custom tool called Hotmap. Frequently requested tiles in the analysis tended to correspond with heavily-populated areas and landmarks such as roads [16], [17]. Drawing upon this research, Guan et al. discuss the process of benchmarking tiled web map applications and propose a custom web map workload that resembles real usage patterns [17]. Providing further analysis on performance, Andreolini et al. investigate tools for benchmarking web applications [18]. Of particular note is the practice of capturing and replaying request logs, which provides a straightforward (albeit limited) method to measure latency during specific request patterns [18].

Response times in web maps can be improved by prefetching tiles that a user is anticipated to request in the future. In [19], García et al. use a neural network trained on prior request patterns for this predictive task. More recent benchmarks found that vector tiles can also provide faster response times and reduced file sizes as compared to raster tiles [20]. However, some datasets (such as SMC) are better-suited for raster formats. Netek et al. found that the newer WebP image format was beneficial for raster tiles [20].

Effective visualizations are predicated on fast data accesses; spatiotemporal data management can be based on sketches [21], DHTs [22], [23], and peer-to-peer grids [24]. PERISCOPE can interoperate with data managed using diverse frameworks.

III. METHODOLOGY

Our methodology encompasses a carefully calibrated set of tasks that together facilitate real-time rendering of voluminous spatiotemporal data. These include: (1) dividing the viewport into dynamically generated tiles that form the primary rendering elements in PERISCOPE, (2) effectively staging and partitioning data, (3) supporting key exploration constructs and a set of rendering refinements that target how tiles are prioritized for rendering, (4) compressing stored and transmitted data, (5) ensuring that the distributed server-side cache works in concert with the client-side cache to improve rendering latencies, (6) leveraging a client's exploration trajectory to inform cache prefetching of tiles, and (7) support for contrastive visualizations.

A. *Tiling* [RQ-1, RQ-2]

Rather than representing map data as one contiguous image, we partition the spatial extent into a grid of *tiles*. Tiling has the advantage of supporting multithreaded operations and concurrency during rendering. This allows the client to request multiple tiles at once and have requests directed in a round-robin fashion to different worker instances. Additionally, this improves responsiveness by allowing the client-side map to update incrementally and asynchronously as tiles arrive.

However, it is not feasible to rely on just one grid of tiles. Rendering all tiles at the highest resolution format regardless of the zoom level would result in: (1) increased disk I/O and data movements, (2) higher computational costs for rendering tiles, and (3) increased costs for network transmissions.

Instead, we rely on perceptual limits. The finest resolution is used only when rendering at the lowest zoom level. We dynamically generate tiles at intermediate resolutions by downscaling and combining finer-grained tiles (Fig. 1). To limit runtime latency, we precompute two of these coarser overview layers and store them alongside the higher-resolution data. Because these tiles are coarser grained, their data volumes are substantially lower. This allows us to conserve data storage, especially since we only compute a limited number of layers.

We rely on two different systems to uniquely reference tiles. First, each tile is assigned a unique coordinate (x, y, z). As tiles become finer-grained, the zoom level z increases. Specifically, the map area covered by a tile decreases by a factor of 4 when z increases by 1.

The second referencing scheme identifies tiles using quadkeys, which consist of digits from 0 to 3. Each successive digit in a quadkey descends a level deeper in zoom, where 0 represents the child tile occupying the upper-left quadrant of its parent and 3 represents the lower-right tile [25]. The length of a quadkey is equal to the z coordinate of its associated tile.

B. *Data Staging* [RQ-2]

We validate our ideas with diverse datasets at different spatial and temporal resolutions. Soil moisture data (daily at

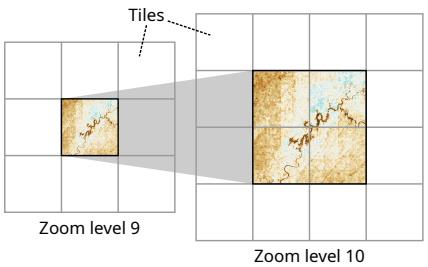


Fig. 1. At all zoom levels, the viewport is divided into multiple tiles. Each successive zoom level decreases a tile's covered area by a factor of 4, and accordingly increases the level of detail captured within.

30m resolution) are computed by deep learning models [26] that are trained on many data sources, most prominently the HydroBlocks project [27]. We also incorporate data from the European Space Agency's Copernicus Digital Elevation Models [2], NASA's Soil-Moisture Active Passive satellite system [3] (36km spatial resolution), and the National Land Cover Database (NLCD) [1]. PERISCOPE renders 30 successive days of soil moisture data from the southwestern United States as a proof-of-concept, with aforementioned datasets provided for comparison and layering.

DISTRIBUTED KEY-VALUE STORE We store raster data tiles as individual GeoTIFFs within an Apache Cassandra distributed hash table database. Unlike relational databases that identify stored items using a single primary key, Cassandra combines a partition and clustering key [28]. The former is hashed to select the partition on the node that stores an item, while the latter uniquely identifies items within partitions [28].

To ensure that spatial locality is preserved for neighboring tiles, we split quadkey identifiers describing tiles into two parts. The last four characters, denoting the most fine-grained position information, are used within the Cassandra clustering key. The rest of the quadkey, describing coarser-grained position, is used within the partitioning key. In the case of time series data, the temporal granularity (month) is also part of the partitioning key, providing temporal locality.

To manage datasets across different spatial resolutions, we associate metadata with each individual dataset. This metadata includes the dataset's spatial bounds, temporal bounds, base zoom level, and any additional precomputed overview levels.

C. *Explorations & Rendering Refinements* [RQ-1]

Spatiotemporal explorations within PERISCOPE encompass the following operators: (1) spatial panning, (2) spatial zooming-in and zooming-out, and (3) temporal drill-downs and roll-ups. Interactivity should be preserved during these operations. Furthermore, a user should be able to perform additional exploration operations that potentially interrupt in-progress loading.

Our tile-based viewport, based on deck.gl [29], allows raster data to be *streamed* as the user performs exploration operations, rather than forcing the user to block waiting for complete, full-screen map rendering. In PERISCOPE, we rely on deck.gl's incremental rendering of streamed tiles to further enhance interactivity. This is accomplished by initially displaying lower-resolution tiles from previously-loaded zoom levels, and having the viewport be incrementally refined during zoom-in operations as higher-resolution tiles arrive.

Our methodology incorporates an additional refinement to ensure that the part of the viewport that the user is most interested in loads first. In particular, we prioritize tile rendering by ensuring that the viewport loads or refines tiles from the center outward. Tiles that overlap the viewport rectangle boundary are loaded last, because those tiles are in the periphery and, being partially cut off, communicate the least visual information.

D. *Compression* [RQ-1, RQ-2]

We store source data for visualizations as GeoTIFF images within Cassandra. We chose this format because, in addition to being able to represent both floating-point and integer data, it is well-supported by libraries (such as rasterio [30] and GDAL [31]) that we leverage. Previously loaded GeoTIFFs are cached within Redis so that users may quickly switch the color maps used for visualization. We further compress these TIFF files using lossless compression, which reduces storage (on-disk) and caching (in-memory) requirements. Also, this data reduction via compression reduces latencies associated with data retrievals from disk and network transmissions.

During visualization, the server side performs an additional refinement as part of delivering color tiles to the client. The server-side loads the TIFF data, applies a colormap (some via [32]), and then compresses these tiles using WebP. Selecting an efficient color format limits space consumed in server- and client-side caches by colorized tile images, and reduces the network I/O bandwidth required by client-server traffic.

Like the GeoTIFF images, the color-mapped WebP images are also cached within Redis to limit the amount of redundant recompression operations that are performed. While it is safe for directly-visualized color images to use lossy compression, terrain tiles must be represented using a lossless format. This restriction for terrain tiles exists because an error of one unit in the most significant byte can cause deviation in height by over six kilometers, in accordance with the multiplier used when decoding the Mapbox format [33].

E. *Caching* [RQ-1, RQ-2]

To minimize duplicate work, reduce the number of I/O operations, and reduce latencies we leverage caching. The caching scheme in PERISCOPE has two components: a smaller one at the client side and a much larger and distributed server-side cache; see (Fig. 2).

CLIENT-SIDE CACHING Caching on the client side reduces the number of requests that need to be sent to the server. This reduction in communications with the server-side reduces I/O costs at the server-side. After a tile is received, it is stored in the client cache, which can hold 10,000 tiles. When a tile cannot be found in the cache (cache miss), the client retrieves that tile from the server side and also caches it. If the cache is full, some existing tiles in the cache are evicted using an approximate least-recently-used (LRU) algorithm.

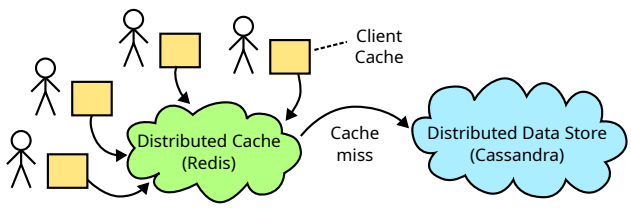


Fig. 2. The clients and server work in concert to maintain client-side and server-side tile caches. In this manner, the number of requests to the backing data store can be minimized, massively reducing application latency.

We preferentially set aside some tiles for residency within the cache. Coarse-grained tiles (zoom level 6) are exempt from cache eviction, because they are stored in a separate client cache where they persist indefinitely. Similarly, coarse-grained tiles from a configurable retrospective window (e.g., 30 days) are preloaded by the client when the application launches, so that temporal drill-downs and roll-ups at this zoom level benefit from reduced latencies.

SERVER-SIDE CACHING We leverage the Redis in-memory storage framework to implement our server-side cache. The Redis and Cassandra systems run on the same set of machines within the cluster.

Our server-side caching scheme accounts for tiles from different zoom levels. To accomplish this, the Redis cluster is organized into four different regions. Each region stores a different range of tiles, segmented by zoom level. Each of these cache regions is treated as a separate "cluster," but they share the same set of machines and are differentiated only by port number. Introducing this separation allows the memory to be separately measured and limited by region, and introduces the possibility of setting differing cache eviction policies.

Tiles at the uppermost overview level (6) are stored within a persistent cache region with no time-to-live (TTL) so that the fully zoomed-out map can be loaded as quickly as possible by clients. Other regions have a TTL configured to expire cached images after one day. Each individual Redis process (there are 4 regions * 62 machines = 248 processes) has a configured memory limit of one gigabyte.

Within each region, tiles are distributed across nodes using a similar partitioning scheme to the Cassandra database: the quadkey prefix of a tile is hashed to determine where it resides. In the case of time-series data, the month number is concatenated and hashed along with that prefix.

SERVER-SIDE DATA STRUCTURES PERISCOPE balances the need to ensure low latencies with space efficiency on the server side. The server-side uses a mixture of offline precomputation and on-the-fly construction of tiles. This is enabled through a specialized cache indexing scheme.

Within the Redis cluster, a radix trie tracks which tiles have been cached. This is enabled by the quadkey indexing scheme, since a parent tile's quadkey will always be a prefix of the quadkeys of its children (e.g. the tile **0320** represents one quarter of the tile **032**).

Using this trie data structure, a recursive loading algorithm minimizes the number of smaller tiles that have to be fetched to compute an overview tile. If a tile is requested that has three-quarters of its children already cached, only the last quadrant needs to be retrieved from Cassandra. Once the server loads all four quadrants, it downscales them by 50% and combines them into a composite tile, which it returns to the client.

F. Visualization trajectories **[RQ-1]**

Visualization trajectories are used to inform prefetching schemes by launching speculative tasks that perform data retrieval operations, cache residency and evictions, and computation of tiles. Our methodology interleaves user dwell times with speculative prefetching of tiles. Dwell times refer to the duration that a user spends visually scanning data and deciding where to pan next.

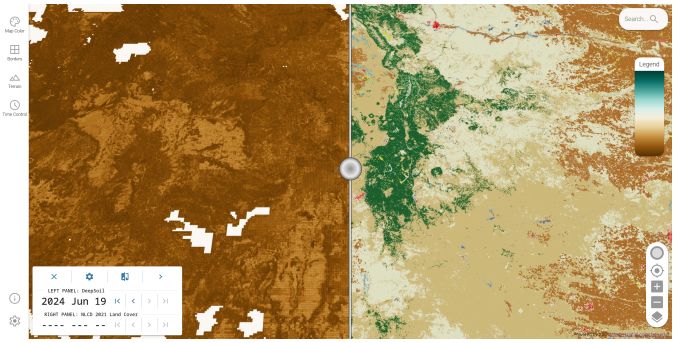


Fig. 3. The PERISCOPE application in use, comparing soil moisture estimates (left) with NLCD 2021 land cover classifications (right).

We encode visualization trajectories and leverage these to predict a user's likely viewport at a future timestep. The visualization trajectory is represented using the following properties: (a) start and end coordinates; (b) delta in longitude and latitude; (c) start and end zoom; (d) viewport bounding box; (e) date; and (f) colormap name.

After the client viewport is moved laterally, zoomed in or out, or advanced forward or backward in time by the user, the application begins prefetching tiles that lie further along the exploratory path. Note that by default for HTTP/1 servers, Google Chrome limits the number of concurrent outgoing requests to 6 per domain [29]. To avoid competing with tiles being loaded for the current map, trajectory prefetching waits until the viewport is fully loaded before it begins issuing requests. Additionally, ongoing prefetching operations are cancelled if the viewport is moved.

G. Contrastive abilities **[RQ-3]**

The frontend provides multiple avenues for contrasting spatiotemporal variation of phenomena. One approach that PERISCOPE supports is to display multiple datasets at once, or one dataset at multiple points in time, using a vertical divider. This is depicted in Fig. 3. In this example, the left half of the screen displays estimated soil moisture data, while the right half displays land cover classifications from the NLCD. Because the dividing slider is draggable, the user can observe the visual difference as an area is rendered using one dataset versus the other.

In addition to having this slider functionality, we support 3D visualizations. This allows soil moisture or other data to be rendered as a texture on top of a 3D terrain mesh. The server encodes elevation data within color image files using a standard encoding scheme as described by Mapbox [33]. Within this encoding, the 8-bit red, green, and blue color channels of each image contain the most significant, middle, and least significant bytes respectively of a 24-bit integer representing elevation. The client, upon receiving this

2D terrain representation, converts it to a 3D mesh using the built-in terrain layer [33].

IV. Performance Benchmarks

We profile several aspects of our methodology. Our empirical benchmarks, experimental setups, and profiling are geared towards answering the following key questions:

- What is the trade-off space for image formats, compression levels, interactivity, and perceptual limits? One of the contributions of this study is a rigorous exploration of this space over voluminous datasets that has the potential to inform similar work in the community. [RQ-1, RQ-2]
- How effective are our caching schemes for rendering at scale? Effective rendering entails the client and server-side caches to work in concert. [RQ-1]
- Do our tiling and data dispersion schemes introduce storage imbalances on the server side? While tiling enables us to ensure responsiveness at the client side, we explore its impact on the server side. [RQ-2]
- How does leveraging exploration trajectories impact latencies during rendering operations? [RQ-1]

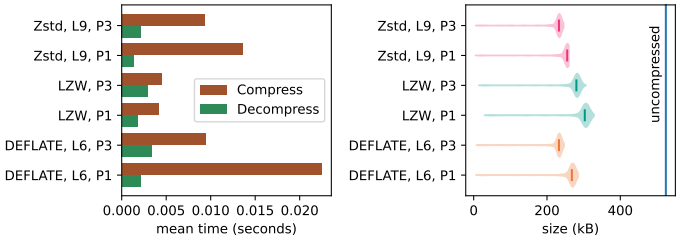
A. Environment

For these benchmarks, the Periscope server components (including the Redis and Cassandra databases) occupied a total of 62 machines running AlmaLinux 8.9 (Linux kernel version 4.18.0). The hardware varied between machines, but the most common configuration had six 2.4 GHz processor cores and 64 GB of RAM. We ran client-side benchmarks in the Google Chrome browser due to its wide use in web environments.

B. Image compression benchmarks [RQ-1, RQ-2]

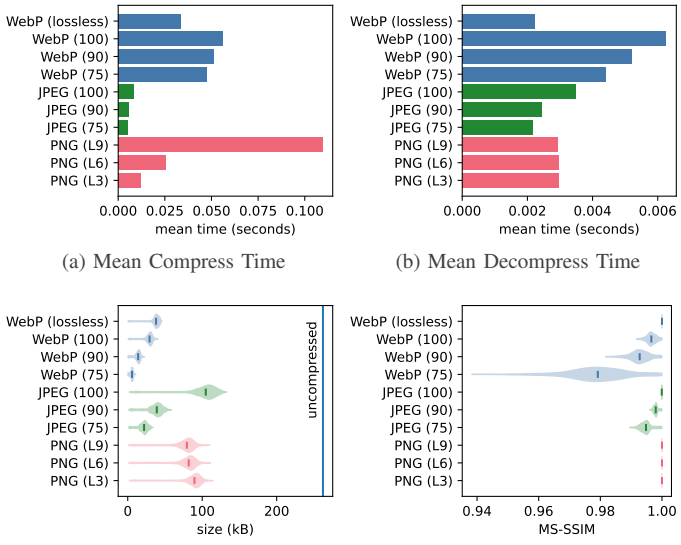
For the following experiments, we randomly sampled 1000 256x256-pixel images from zoom level 10 of the soil moisture data (the first overview level). These tiles were then compressed using different algorithms.

GeoTIFF compression benchmarks Base and overview tiles are stored as lossless-compressed GeoTIFF images in Cassandra. In testing, we found that DEFLATE and Zstandard compression with a floating-point predictor (in GDAL, this is predictor mode 3) resulted in the smallest files. LZW had the shortest compression time, followed by Zstandard. Ultimately, we chose to compress our raw data using Zstandard because of its observed mix of compressed sizes and shorter compression times.



(a) Mean Compression Time (b) Compressed Image Size

Fig. 4. In TIFF image compression benchmarks, Zstandard compression showed better decompression time and file size.



(a) Mean Compress Time (b) Mean Decompress Time

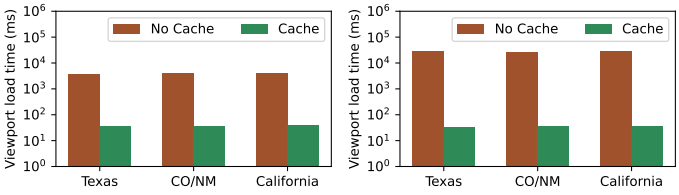
(c) Compressed Image Size (d) Multiscale Similarity

Fig. 5. In color image compression, lossy WebP showed higher time cost, lower MS-SSIM, and much smaller file sizes.

Color image compression benchmarks Fig. 5d shows the measured multiscale structural similarity (MS-SSIM) metrics from various compression algorithms, plotted using [32]. Notably, lossy WebP actually saw the lowest structural similarity out of the formats tested here, and was the slowest out of the lossy formats (Fig. 5a). However, Fig. 5c shows that lossy WebP also produced the smallest output files. Though it is not without tradeoffs, another major advantage of lossy WebP is that, unlike JPEG, it supports RGBA color (with the fourth channel denoting per-pixel transparency). This allows the visualization to be rendered on top of a basemap that provides a stronger spatial reference to users. Lossless WebP also fared well, giving file sizes half those seen with PNG (Fig. 5c). For this reason, we decided to use lossless WebP when rendering tiles at the base zoom level (14).

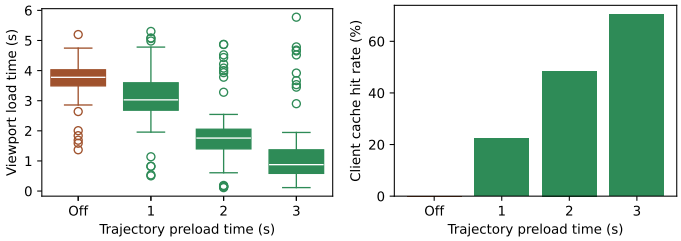
C. Client cache benchmarks [RQ-1]

In order to benchmark the efficacy of the client cache we performed a set of experiments designed to isolate tiles coming from the client cache versus those coming from the server. We implemented a testing harness on the client that started a timer when a request to populate the viewport with tiles went out, and stopped the timer once all tiles were rendered. Each experiment included the following steps: (1) Clear the client cache; (2) Start the viewport at a specific zoom level; (3) Move the viewport left by 1 viewport width, timing how long it takes for all tiles to render; (4) Repeat step 3 ten times; (5) Move the viewport right by 1 viewport width, timing how long it takes for all tiles to render; (6) Repeat step 5 ten times.



(a) Zoom level 11 (b) Zoom level 9

Fig. 6. Semi-log plots showing time to load viewports with and without caching. Caching decreased rendering times by orders of magnitude.



(a) Viewport load time (b) Cache hit rate

Fig. 8. Trajectory preloading improved both load time and cache hit rate.

When moving the viewport left with a clear client cache, there are 0 cache hits. We confirmed this by logging all cache hits to the console. When moving the viewport right, the cache hit percent was 100% because all tiles had recently been fetched/cached. We performed this experiment three times in different regions (Central Texas, CO/NM Border area, California) at two different zoom levels (9 and 11). We used two zoom levels to account for the fact that different zoom levels relate to different tile sizes, and different numbers of tiles. The results are depicted in Fig. 6a and 6b.

Using a client cache dramatically reduces the time to render all requested tiles. In the case of zoom level 11, the rendering times were reduced by two orders of magnitude. At zoom level 9, the rendering times were reduced by three orders of magnitude. Furthermore, although the time to render tiles without the cache increases by an order of magnitude from zoom level 11 to zoom level 9, there isn't a remarkable difference in time to render these tiles between zoom levels when the tiles are in the client cache.

D. *Profiling server-side data dispersion* [RQ-2]

Fig. 7 shows the amount of space taken up by the soil moisture dataset on each of the nodes comprising our distributed Cassandra database. The disk usage ranges from 18 to 38 gigabytes, so it is not completely uniform. Nonetheless, the distribution does not suggest any hotspots that completely dominate tile apportionment. Because the Redis cache cluster uses a similar hashing scheme to determine tile placement, and includes the same tile properties within the hash keys, its distribution would have a similar pattern.

E. *Exploration trajectory benchmarks* [RQ-1]

We used a similar approach to the cache benchmarking to measure the impact of exploration trajectory prefetching on visualization performance. To take these measurements, we iteratively moved the viewport horizontally back and forth across a large, rectangular region spanning the western United States. When reaching an edge of the rectangle, the viewport would move one step south in latitude and begin moving back the other way in longitude such that its path did not self-intersect. Since this path formed a grid of 15 unique longitudes and 5 latitudes, each benchmark run included 75 different viewport positions at zoom level 10.

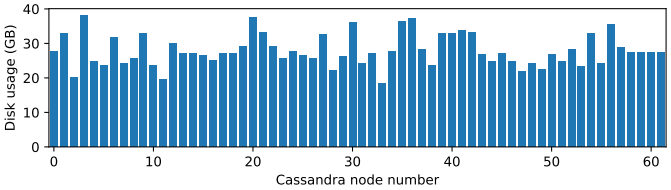


Fig. 7. Disk space used by the soil moisture dataset on each of 62 Cassandra nodes, while varied, does not indicate presence of major hotspots.

The number of tiles that are loaded by a prefetching operation is partially determined by the amount of time the user keeps the view stationary, which we refer to as *dwell time*. To account for this factor, the simulated user waited a configurable number of seconds after each viewport move, thereby allowing tiles to preload. The cache was cleared before each benchmark run, ensuring that overview tiles would need to be dynamically computed.

As can be seen in Fig. 8a, the relationship between dwell (preload) time and viewport load time is roughly linear. If a simulated user waited one second after the current viewport loaded, then the time to load the next viewport was reduced by an average of around 0.75 seconds from how long it would otherwise take. At a dwell time of 3 seconds, the next viewport would load in roughly 1 second.

This effect is further seen in the client cache hit rate (Fig. 8b). With no prefetching, the cache hit rate is zero due to the non-self-intersecting path. With prefetching at a dwell time of one second, the cache hit rate rises to 20%. At two seconds, the hit rate is 50%, and three seconds reaches 70%.

This benchmark captures the case where viewport movement is almost completely linear, so we expect some variability in real-world performance. Nonetheless, trajectory prefetching in PERISCOPE does not start until the user's current viewport finishes loading, and is cancelled when the viewport is moved again. As a result, our methodology ensures that PERISCOPE will not compete for client bandwidth against the tiles that are actually visible (unless the user's connection is metered).

V. CONCLUSIONS & FUTURE WORK

Here we described our methodology to render spatiotemporally evolving phenomena interactively and at scale.

RQ-1: Leveraging tiling allows us to partition the viewport as a collection of tiles. Combining tiling with preferential caching, incremental refinement, and streaming allows us to

preserve interactivity. Ensuring that the cache accounts for usage patterns and exploration trajectories allows our cache hit rates to be high which, in turn, benefits explorations. In PERISCOPE, caches at the client and server-side work in tandem to ensure reduced I/O at the server side alongside reduced network transmissions.

RQ-2: Managing the competing pulls of dispersion and data locality using quadtiles allows us to collocate observations from a spatial extent at the same machine allowing for batched transfers that are far more efficient than smaller, sporadic transfers. The use of quadtiles and hashing-based dispersions allows the data collections to dispersed across the collection of machines. This load balancing allows the server-side to avoid hotspots in data storage. Maintaining a distributed, server-side cache allows us to exploit usage patterns across a collection of users. This allows the most popular tiles to be memory-resident. Supplementing this with preferentially cached tiles allows the server-side to warm-start visual explorations at the clients. Finally, data are compressed both at rest (on-disk) and in motion (transmissions) – this substantially reduces the amount of I/O that needs to be performed.

RQ-3: Our rendering of spatiotemporally evolving phenomena are based on constructs such as caching, tiling, and compression. These abstractions translate very well across different resolutions allowing us to effectively support them. Our contrastive capabilities rely on bisecting the viewport that is amenable to pinch-and-drag operations. Together the panes represent a contiguous, underlying spatial extent that is preserved during the bisection drag operations. Intuitive timing controls allows users to overlay different datasets from different timesteps in each pane. Temporal advancements in the different panes can be performed in lockstep.

In our future work we will explore leveraging models to render phenomena. In particular, the goal is to seed models with a limited amount of data and have these models generate the tiles at high fidelity. We expect that this should significantly alleviate data transmission requirements. We also plan to improve accessibility in the visualization engine by supporting diverse color palette schemes for color discernment impairments.

REFERENCES

[1] J. Dewitz, "National Land Cover Database (NLCD) 2021 Products," 2023. [Online]. Available: https://doi.org/10.5066/P9JZ7AO3

[2] European Space Agency, "Copernicus Global Digital Elevation Model," 2024. [Online]. Available: https://doi.org/10.5069/G9028PQB

[3] P. E. O'Neill, S. Chan, E. G. Njoku, T. Jackson, R. Bindlish, J. Chaubell, and A. Colliander, "SMAP L2 Radiometer Half-Orbit 36 km EASE-Grid Soil Moisture, Version 8," 2021. [Online]. Available: https://nsidc.org/data/SPL2SMP/versions/8

[4] "Business Intelligence and Analytics Software | Tableau." [Online]. Available: https://www.tableau.com/

[5] "Spotfire: Solving complex, industry-specific problems at the speed of thought." [Online]. Available: http://www.spotfire.com

[6] "Statistical Software | JMP." [Online]. Available: https://jmp.com

[7] "NetCharts Performance Dashboard and Data Visualization Software." [Online]. Available: https://visualmining.com/

[8] J.-D. Fekete, "The InfoVis Toolkit," in *IEEE Symposium on Information Visualization*, Oct. 2004, pp. 167–174, iSSN: 1522-404X.

[9] J. Heer, S. K. Card, and J. A. Landay, "prefuse: a toolkit for interactive information visualization," in *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*. USA: ACM, Apr. 2005, pp. 421–430.

[10] C. Weaver, "Building Highly-Coordinated Visualizations in Improvise," in *IEEE Symp. on Information Visualization*, Oct. 2004, pp. 159–166.

[11] J. J. Thomas and K. A. Cook, *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Center, 2005.

[12] Z. Liu, B. Jiang, and J. Heer, "imMens: Real-time Visual Querying of Big Data," *Comput. Graph. Forum (Proc. EuroVis)*, vol. 32, no. 3, 2013.

[13] C. A. L. Pahins, S. A. Stephens, C. Scheidegger, and J. L. D. Comba, "Hashedcubes: Simple, Low Memory, Real-Time Visual Exploration of Big Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 671–680, Jan. 2017.

[14] H. Liu and Y. Nie, "Tile-based map service GeoWebCache middleware," in *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, vol. 1, 2010, pp. 692–697.

[15] H. Wu, X. Guan, T. Liu, L. You, and Z. Li, *A High-Concurrency Web Map Tile Service Built with Open-Source Software*. Boston, MA: Springer US, 2013, pp. 183–195.

[16] D. Fisher, "Hotmap: Looking at geographic attention," *IEEE Trans. on Visualization and Comput. Graph.*, vol. 13, no. 6, pp. 1184–1191, 2007.

[17] X. Guan, B. Cheng, A. Song, and H. Wu, "Modeling users' behavior for testing the performance of a web map tile service," *Transactions in GIS*, vol. 18, no. S1, pp. 109–125, 2014.

[18] M. Andreolini, V. Cardellini, and M. Colajanni, "Benchmarking models and tools for distributed web-server systems," in *Performance Evaluation of Complex Systems: Techniques and Tools*, M. C. Calzarossa and S. Tucci, Eds. Springer Berlin Heidelberg, 2002, pp. 208–235.

[19] R. García, E. Verdú, L. M. Regueras, J. P. de Castro, and M. J. Verdú, "A neural network based intelligent system for tile prefetching in web map services," *Expert Systems with Applications*, vol. 40, no. 10, pp. 4096–4105, 2013.

[20] R. Netek, J. Masopust, F. Pavlicek, and V. Pechanec, "Performance testing on vector vs. raster map tiles—comparative study on load metrics," *ISPRS Int. J. of Geo-Information*, vol. 9, no. 2, 2020.

[21] T. Buddhika, M. Malensek, S. L. Pallickara, and S. Pallickara, "Synopsis: A distributed sketch over voluminous spatiotemporal observational streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2552–2566, 2017.

[22] M. Malensek, S. Pallickara, and S. Pallickara, "Evaluating geospatial geometry and proximity queries using distributed hash tables," *Computing in Science & Engineering*, vol. 16, no. 4, pp. 53–61, 2014.

[23] S. Mitra, P. Khandelwal, S. Pallickara, and S. L. Pallickara, "Stash: Fast hierarchical aggregation queries for effective visual spatiotemporal explorations," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2019, pp. 1–11.

[24] G. Fox, S. Pallickara, and X. Rao, "Towards enabling peer-to-peer grids," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 7-8, pp. 1109–1131, 2005.

[25] R. Brundritt, S. Munk, C. French, and S. Cai. (2024) Bing Maps Tile System. [Online]. Available: https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system

[26] P. Khandelwal, S. L. Pallickara, and S. Pallickara, "DeepSoil: A Science-guided Framework for Generating High Precision Soil Moisture Maps by Reconciling Measurement Profiles Across In-situ and Remote Sensing Data," in *The 32nd ACM Int. Conf. on Advances in Geographic Information Systems (SIGSPATIAL '24)*, Atlanta, GA, USA, 2024.

[27] N. W. Chaney, P. Metcalfe, and E. F. Wood, "HydroBlocks: a field-scale resolving land surface model for application over continental extents," *Hydrological Processes*, vol. 30, no. 20, pp. 3543–3559, 2016.

[28] "Introduction | Apache Cassandra Documentation," 2024. [Online]. Available: https://cassandra.apache.org/

[29] deck.gl contributors, "TileLayer." [Online]. Available: https://deck.gl/docs/api-reference/geo-layers/tile-layer

[30] S. Gillies *et al.*, "Rasterio: geospatial raster i/o for Python programmers," Mapbox, 2013–. [Online]. Available: https://github.com/rasterio/rasterio

[31] GDAL/OGR contributors, *GDAL/OGR Geospatial Data Abstraction software Library*, Open Source Geospatial Foundation, 2024. [Online]. Available: https://gdal.org

[32] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[33] deck.gl contributors. TerrainLayer. [Online]. Available: https://deck.gl/docs/api-reference/geo-layers/terrain-layer