CADDY: Scalable Summarizations over Voluminous Telemetry Data for Efficient Monitoring

Saptashwa Mitra, Scott Ragland, Vanessa Zambrano, Dipanwita Mallick, Charlie Vollmer, Lance Kelley, Nithin Singh Mohan
HPC and Slingshot Business Unit, Hewlett Packard Enterprise
{saptashwa.mitra, scott.ragland, vanessa.zambrano, dipanwita.mallick, charlie.vollmer, lance.kei.kelley, nithin-singh.mohan}@hpe.com

***Abstract*—In the rapidly evolving landscape of High-Performance Computing (HPC), the efficient management and analysis of telemetry data is pivotal for ensuring system robustness and performance optimization. As HPC systems scale in complexity and capability, traditional data processing methodologies struggle to meet the demands of rapid real-time analytics and large-scale data management. This paper introduces an innovative framework, Caddy, which employs a novel approach to HPC telemetry storage and interactive analysis. Built on the foundation of HPE's Slingshot interconnect and the Fabric AIOps (*FAIO*) system, Caddy aims to address the critical need for a memory-efficient, scalable, and real-time analytical solution for seamless monitoring over large HPC environments.**

Fabric AIOps' reliance on voluminous telemetry data generated from Slingshot's nodes and switches poses significant challenges for traditional disk-based storage solutions and their ability to efficiently parse, analyze, extract real-time insights, identify potential bottlenecks, and ensure seamless operation. Telemetry analytics, particularly in cases of aggregation over large sections of the fabric, involves high disk I/O, network transfers, and record processing. Their interactivity and consistency are further impeded by constantly streaming telemetry and multiple simultaneous analytical queries.

In-memory analytics can offer significant speed-up over disk-based approaches and enable significantly accelerated aggregation computations, but limited memory capacity at scale remains a critical hurdle for traditional caching. This paper explores Caddy, a novel in-memory storage and summarization framework, emphasizing its ability to generate and accurately apply dynamic updates to *low-dimensional representations of telemetry data* as well as provide quick data access during query evaluations. Caddy facilitates instant insights, minimizes memory footprint, and retains accuracy in analytical processes, even under intense HPC demands.

System benchmarks over Caddy demonstrate that a Caddy-enabled FAIO live mode reduces query latency of a single-frame by *∼2x* for a switch-level telemetry aggregation. Caddy also reduced the memory footprint of in-memory telemetry drastically – for instance, Bins with temporal sizes of 10, 15, and 30 mins, the compression factors achieved were *∼425x, 600x, and 1200x*, respectively,

I. INTRODUCTION

High-Performance Computing (HPC) environments are data-intensive systems, characterized by high-speed interconnects for rapid large-scale data communication between compute nodes. To ensure system health and performance, such as, detailed telemetry data is collected from these interconnects, monitoring metrics like link utilization, packet latencies, and error rates. In *exascale* HPC clusters with thousands of nodes and switche, and constant communication, complex interconnect fabrics, with each channel generating a continuous stream of data, the telemetry volume can rapidly grow to the order of petabytes over a short period of time.

This telemetry data, containing detailed information about the state of the interconnect fabric, can play a critical role in ensuring optimal performance of the system by facilitating efficient monitoring, debugging, and enabling adaptive resource management. By leveraging advanced analytics techniques on this high-volume and high-velocity telemetry data, HPC users and system administrators can pinpoint performance bottlenecks, expedite diagnostic processes, and implement adaptive resource management strategies.

The Slingshot interconnect [15] is an essential component of HPE supercomputer systems, offering comprehensive network performance, monitoring capabilities with real-time data analysis. Fabric AIOps is the analytical framework built upon Slingshot's monitoring APIs that empowers system administrators and researchers with visibility into network performance, job execution efficiency, and communication patterns. Fabric AIOps' aims to simplify and optimize access to Slingshot telemetry data through APIs, enabling users to seamlessly explore and analyze network performance for enhanced system monitoring and in-depth analysis for long-term system maintenance.

However, while frameworks exist to manage the entire telemetry data lifecycle – from collection and storage – the storage layer itself presents a significant bottleneck to the extraction of fast analytics and utilization of this underlying data for *extraction of meaningful insights* regarding the fabric in a scalable manner. Disk-based persistent storage systems introduce I/O overhead that hinders the responsiveness of real-time analytics needed for effective HPC telemetry management. This latency is further exacerbated by the fact that analysis over large time ranges for fabrics of exascale systems entails processing massive amount of datapoints. This *lag* leads to reduced responsiveness of the user interface that can hamper the usability of the product [16].

In-memory computing offers a transformative approach by enabling significantly faster data access and manipulation compared to disk-based storage. For large datasets, distributed

in-memory systems have demonstrated significantly improved query latency (in the order of 100x) compared to distributed disk-based storage systems [3]. However, a key challenge lies in scaling these in-memory storages to accommodate voluminous, ever-growing datasets, as they're ultimately limited by the individual memory capacity of the host node(s).

A. Challenges

Rapid ingestion and analytics over high-volume telemetry, especially over large temporal ranges, for the identification of patterns or diagnostics over the fabric entails several challenges:

1) **Data volume**: Telemetry data collected at high-frequency over exascale network fabrics is voluminous. Inefficient storage systems do not scale well and would entail significant latency for analytical queries in identifying relevant data segments and processing them to extract requested metrics.
2) **Limited Memory Capacity**: Due to the sheer number of data sources in exascale HPC systems, the amount of generated telemetry data over which we can perform rapid summarizations is limited by the capacity of the machine's in-memory storage. Our in-memory store, if not optimized, would drastically reduce the amount of recent telemetry data we can perform rapid explorations over. Additionally, with constantly streaming telemetry measurements being ingested into the live mode, supporting the fast retrievals over recent telemetry requires implementation of an efficient eviction scheme for in-memory data objects.
3) **Serving Concurrent Queries Over Varying Granularities**: Users may request aggregations and summarizations at diverse scopes, i.e., over arbitrary temporal and telemetry bounds. The telemetry bounds can be based on specific segments of the fabric, identifier by their group/switch/port numbers. Redundant evaluations over data segments for these user queries are common and if not specifically handled, can significantly balloon the utilization of resources in our system and increase lag.

B. Research Questions

In order to enable sysadmins to summarize and aggregate over telemetry data at scale over an in-memory data store, the following are specific research questions that we neeed to explore:

RQ-1: How can we scale with increases in data volumes from continuously streaming telemetry data generation?

RQ-2: How can we ensure that our optimized data model (for more efficient in-memory storage) does not compromise on accuracy of aggregation results in the face of continuously streaming data?

RQ-3: How do we efficiently orchestrate in-memory storage of high-velocity data to ensure that the most recent telemetry data is always resident in our distributed in-memory storage?

RQ-4: How can we preserve interactivity of response for summarizations across arbitrary scopes?

C. Overview of Approach

In this paper, we propose an optimized in-memory data store, **Caddy**, built on top of Fabric AIOps' *live mode*, which is a lightweight Ray-based [17] in-memory store. Caddy is specifically designed to significantly *boost the live mode's storage capacity* as well as improve query times over the in-memory telemetry data. Live mode is an auxilliary data-storage mode in FAIO that allows analytics over *10 minutes* of the most recent telemetry data in an interactive manner. Overall, by enabling the storage of a significantly larger segment of the recent fabric telemetry in-memory and faster evaluations through its novel data model, Caddy enables users and sysadmins to extract real-time insights. We allow viewing of the fabric over a diverse range of temporal and fabric resolutions through a playback feature of a sequence of aggregated snapshots of telemetry statistics.

Caddy's data model is designed to cater specifically to the requirements of HPC telemetry data collected over large interconnect fabrics, taking into consideration the relationships between the fabric components of the HPC system. The following highlights some of the key aspects Caddy aims to address:

High-Throughput Data Ingestion: Telemetry data is generated at high velocity, particularly in large-scale HPC deployments. Our in-memory data store is designed to efficiently ingest this data stream, minimizing memory-footprint and ensuring accurate, real-time capture of critical information and patterns.

Efficient Data Management and Accuracy: We optimize the telemetry data management to go beyond simply storing of compressed data and incorporate statistical methods for efficient data organization, indexing, and compaction of telemetry without compromising on their accuracy. We ensure efficient utilization of memory resources while facilitating rapid data retrieval for analytical queries through a metadata-driven indexing scheme.

Scalability: Interconnects, specifically in case of exascale systems are enourmous in scale, both in terms of size and complexity. Our in-memory data store is designed to scale seamlessly to accommodate growing data volumes and evolving HPC needs. This scalability is achieved through a configurable compression scheme that breaks up the overall data domain into equal smaller sub-domains and maintaining aggregate statistics with constant memory footprint. We also allow for configurable sliding time window over which telemetry data is housed in-memory depending on the scale of the fabric.

Fast Data Retrieval and Analysis: Caddy amplifies FAIO's ability to retrieve and analyze telemetry data rapidly. Our in-memory storage allows for significantly faster data access compared to traditional storage solutions through implementation of a hierarchical indexing scheme, enabling

real-time analysis and fostering a more responsive monitoring of the HPC environment.

Integration with Existing Workflows: Caddy is well-designed to integrate effortlessly with existing FAIO workflows. Our solution is designed to seamlessly interact with existing telemetry data management frameworks and analysis of FAIO, minimizing disruption to established workflows.

By reducing reliance on persistent storage, our solution simplifies system administration and maintenance tasks. In conclusion, this paper proposes a novel in-memory data store specifically designed for telemetry data management in large-scale HPC environments. Our solution aims to address the limitations of traditional in-memory storage solutions by leveraging knowledge of the fabric topology in the design of the data model for better indexing of data objects. By optimizing data ingestion, retrieval, and analysis, our in-memory data store has the potential to revolutionize the way telemetry data is utilized for diagnostics within HPC workflows, ultimately leading to a more performant, responsive, and efficient HPC ecosystem.

Our benchmarks demonstrate that a live mode enabled with our Caddy framework reduces query latency of a single-frame (a small scale telemetry-query) by *2x* for a switch-level telemetry aggregation, showing the efficiency of Caddy's storage and indexing scheme. Caddy also reduced the memory footprint of in-memory telemetry drastically. For Bins with temporal sizes of 10, 15, and 30 mins, the compression factor for our in-memory storage were *∼425x, 600x, and 1200x*, respectively, showing that adopting this framework for the storage of telemetry can be greatly beneficial for diagnosing an interconnect's fabric over long ranges of times (in the order of days) in an interactive manner. Additionally, our maintenance of framework shows only a 32.5% overhead in ingestion speed of telemetry into our in-memory storage.

D. Paper Contributions

The following outline the contributions of Caddy to the functionality of FAIO's live mode:

1) An optimized, scalable in-memory framework that supports continuous assimilation of streaming telemetry data, targeted I/O, and cache-residency schemes to minimize redundant processing in a multi-user system. This replaces the current live mode in-memory storage framework of FAIO.
2) Interactive summarizations over varying data granularities for the users to be able to view the facric at varying resolutions for easy exploration and diagnosis of the fabric.
3) Efficient evaluation schemes over our distributed cache to alleviate disk access and avoid re-computation costs.
4) A configurable framework to support any number of telemetry counters/variables collected from the fabric. We allow aggregations over attributes over any requested subsection of the fabric and time range.

Translational Impacts: In its current state, Fabric AIOps, due to the rate of generation of telemetry data, combined with the size of the fabric of some of the HPC systems that it serves, is restricted by the incoming data volume and can accomodate only a small window of latest telemetry in-memory for its *live mode* analytics (10 minutes, traditionally). Through Caddy's optimized storage model and it's leveraging of statistical methods to ensure accurate representation of in-memory aggregates over its underlying telemetry, we aim to alleviate this limitation of FAIO's live mode and bringing up the live mode's storage capacity to the order of a few days' worth of telemetry in-memory. We also take into consideration that our in-memory enhancements has minimal maintenance overhead and does not have a serious adverse effect of the rate of data ingestion for our system.

E. Paper Organization

The remainder of this paper is organized as follows. Section II outlines background and related works, followed by Section III that briefly describes tha various components of the FAIO architecture and outlines where Caddy fits in. Section IV describes the Caddy data model and the construction and the hierarchical accumulation of Caddy data units. Section V details our experimental setups, performance benchmarks, and analysis of the results. Finally, Section VI outlines our conclusions, followed by possible future scopes in Section VII.

II. Background and Related Work

Caddy addresses the usecase of an HPC system administrator tasked with ensuring the health of a complex, interconnected fabric, like the Slingshot interconnect. Diagnosing potential issues, failures, or bottlenecks often requires revisiting historical telemetry data for a significant temporal window, such as the preceding few hours or even days. Traditional storage solutions would involve retrieving and analyzing large telemetry datasets from disk-based database storage delaying critical troubleshooting efforts. Caddy provides an optimized data model that supports a high-fidelity fabric telemetry playback feature for expedited HPC troubleshooting. By storing compressed fabric telemetry data in RAM, near-instantaneous retrieval of historical information becomes possible at varying resolutions. This translates to a user interface (UI) enabling seamless *rewinding* of fabric activity over a desired timeframe for the system administrators to browse the network behavior. This swift access to historical data facilitates expedited diagnoses and more efficient troubleshooting within the HPC environment. Fig.1 shows a snapshot of Caddy's UI that allows sysadmins to view a series of snapshots of the fabric's telemetry in sliding windows of 10 mins in order to provide them with an understanding of its status over the last 24 hours.

Scalable visual analytics [8], [9] of voluminous datasets requires overcoming query latency bottlenecks. Prefetching or pre-aggregation techniques that enable fast query execution over voluminous datasets are a commonly used technique for optimizing data analysis workloads when dealing with large datasets and frequent queries that involve aggregations over varying granularities. It offers an effective strategy to reduce

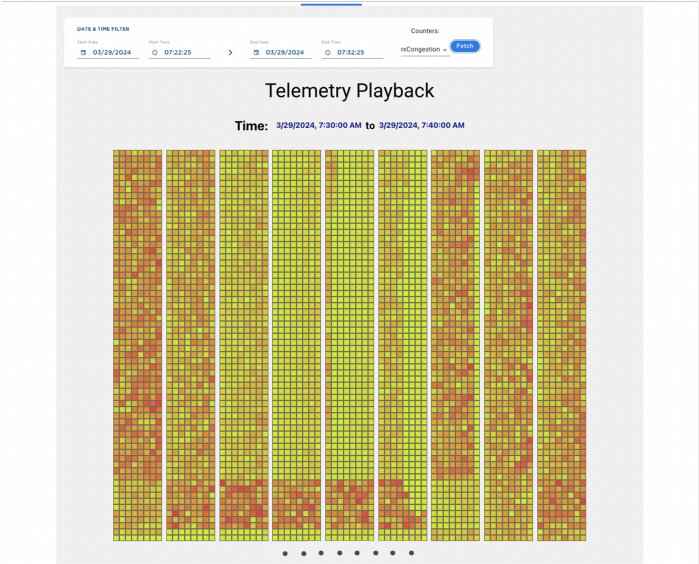

Fig. 1: **Caddy User Interface**: Visualization of telemetry in consecutive timeslices in an interactive mannes (telemetry playback/rewind)

memory footprint and accelerate query execution on large datasets by reducing computational overhead.

In the field of aggregated storage and analytics, tile layers have commonly been used to facilitate multi-resolution aggregate analytics [6]–[8]. These smaller units hold pre-computed aggregations (SUM, COUNT, etc.) at varying levels of granularity (spatial, temporal, etc.) [4], [5]. By organizing these data units (tiles) in a hierarchical fashion, queries can directly access these pre-aggregated segments based on the requested granularity for faster retrieval, reducing memory usage and query execution time. This also alleviates the need for redundant processing the entire raw data. Additionally, the data tile layer is configurable and can be optimized based on common queries and access patterns, ensuring efficient utilization of storage resources and accelerating insights discovery [10].

Traditional approaches to storing raw data can become I/O bound, hindering performance in HPC environments [19], [20], especially in cases of data generated by exascale systems. Several research efforts have demonstrated the advantages of in-memory distributed storage for accelerating large-scale analytics. This is particularly true in case of platforms built for scalable analytics over High-Performance Computing (HPC) environments [11]. Analytics frameworks supporting in-memory persistence schemes [12] support direct fetch of data-elements between successive operations and have demonstrated speedups of upto 100 folds in terms of query latency for queries compared to that of disk-based large-scale analytical frameworks. Intuitively, this performance boost is due to the reduced latency associated with in-memory data access compared to disk I/O operations, highlighting the potential of in-memory housing of telemetry for HPC analytics workloads.

III. Overview of Architecture

FAIO is essentially a data analysis framework designed to bridge the gap between raw telemetry data collection from the Slingshot interconnect and extraction of actionable insights for High-Performance Computing (HPC) systems [14]. It extends the features of existing monitoring frameworks such as Cray System Manager and HPE Performance Cluster Manager by offering a more diverse and detailed analysis of the fabric characteristics.

We outline the key components of the FAIO framework in Fig.2 and give brief descriptions of each of these components below.

ETL Layer: This layer acts as an adapter, fetching data from the Slingshot telemetry APIs exposed by the existing monitoring frameworks. It performs post-processing for raw telemetry data into a standardized format for efficient storage and querying over the FAIO system.

Analytics Data Store Layer: This layer is responsible for combining and storing network fabric performance data in a format optimized for analytics. This involves data transformation and aggregation techniques for faster and more efficient querying. FAIO offers two data storage modes catering to distinct use cases. The **historical mode** utilizes persistent on-disk storage solutions, such as TimescaleDB [21] or OmnisciDB [22], optimized for long-term archival and retrieval of network performance data. This mode caters to *long temporal callbacks*, enabling analysis of historical trends, for example, viewing detailed fabric telemetry aggregates and patterns from months back. In contrast, the **live mode** employs a lightweight in-memory storage for the most recent telemetry data (typically a few minutes). This facilitates rapid analytics on the current network state for live diagnostics.

Caddy enhances the FAIO's live mode capabilities. Through

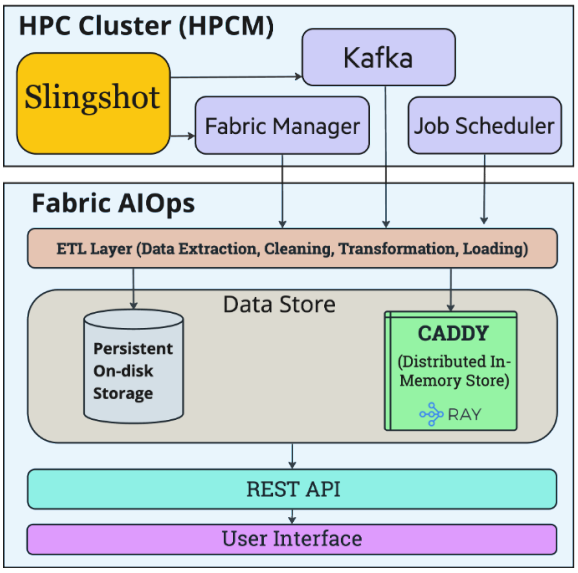

Fig. 2: Fabric AIOps Architecture

its integration with the live mode, Caddy optimizes the data model for efficient storage and querying. By employing hierarchical aggregation techniques, Caddy significantly increases the live mode's in-memory storage capacity to support several days of data. This extended window facilitates interactive diagnostics and analysis of fabric health and behavior over a longer timeframe.

FAIO API Layer: This layer offers a set of Python calls specifically designed for performing complex data analysis tasks on the network performance data. These calls include requests tasks like filtering, aggregation over the whole or sections of the Slingshot fabric.

RESTful API Layer: This layer provides a RESTful API (accessible through standard web requests) for the user interface that allows users to interact with FAIO's functionalities. Users can request aggregations over different time ranges or fabric subsection, visualize utilization of resources for jobs and highlight the span of specific jobs over the fabric, among others, through various interactive visual representations for deeper insights.

To summarize, FAIO streamlines the process of extracting meaningful insights from network performance data by simplifying data acquisition, and optimizing storage for faster analysis. It offers powerful analytical tools, and provides a user-friendly interface for interaction allowing researchers and system administrators to gain quicker and deeper understanding of their HPC system's network behavior, leading to improved network monitoring and maintenance.

IV. Caddy Data Model for Scalable Analytics

Our primary goal is to enhance memory efficiency in handling large-scale telemetry data of exascale systems, ensuring that larger volumes of in-memory data can be analyzed in real-time without sacrificing accuracy. This improvement is crucial for system administrators who rely on *timely* and *precise* data to monitor and manage high-performance computing environments. Caddy optimizes the Fabric AIOps' *live mode* (Fig. 2), which is an auxiliary distributed lightweight storage that allows real-time network health monitoring over a small segment of the most recent telemetry (10 minutes' worth). The key feature of Caddy's novel design is its ability to rapidly compute and store aggregates in an online manner at varying granularities during data ingestion to serve multi-resolution analytical queries, while maintaining a consistent memory-footprint for each data aggregate, irrespective of data density.

A. Telemetry Bins

Caddy enables efficient management of high-volume, high-velocity data by employing a *tile layer* [13] model for in-memory aggregation. We summarize raw data into compressed, hierarchical tiles at varying granularities. Caddy groups and aggregates incoming telemetry data into discrete bounds based on their telemetry and temporal metadata. Instead of storing individual telemetry data-points, we group all data-points over a sub-domain and represent them with a single set of values. This reduces redundancy and storage requirements significantly by keeping the data size for each sub-region constant, irrespective of the volume of the underlying telemetry. This approach is critical for handling massive data streams from the Slingshot interconnect fabric because traditional methods would quickly overwhelm storage systems and slow down analytics. By pre-aggregating data, Caddy ensures economical storage and enables faster queries for real-time insights.

A Caddy **Bin** serves as the unit of data aggregation and analysis. These Bins are crucial to encapsulating aggregated values over any configurable set of telemetry measurements (counters). Each Bin represents aggregate

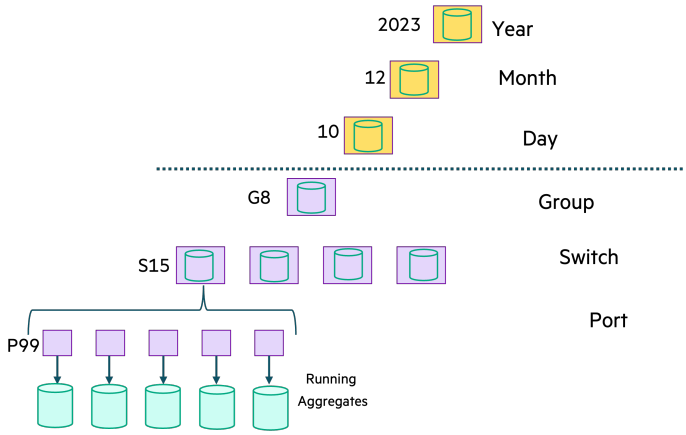

Fig. 3: **Caddy Data Model**: Hierarchical Organization of Bins

analytics over telemetry data for any *configurable range of telemetry (R_f) and temporal resolution (R_t)* of all incoming telemetry events in a stream that fall with the bounds of the Bin. Telemetry resolutions can be at the level of either Group, Switch, or Port while temporal resolution can be Monthly, Daily, or Hourly. The aggregate statistics supported by our system currently include mean, median, variance, skewness, and kurtosis for each of the following 4 counters: **`rxBW`, `txBW`, `rxCongestion`,** and **`txCongestion`**. The **bounds** of each Bin is denoted by the tuple $\{time\_range, group\_id, switch\_id, port\_range\}$, which can be adjusted based on the resolution represented by the Bin which also determines its position in the hierarchy. All events are tagged to their specific Bin based on their metadata and used to update the aggregated contents of that Bin. Bins' aggregate statistics can be updated in an online fashion and kept memory efficient through the use of Welford's online algorithm [18]. Additionally, Bins can be efficiently and accurately merged based on the same algorithm.

For distributed, multi-node storage, Caddy allows for configurable partioning of the in-memory store *partitioned* seamlessly across multiple nodes. In our current implementation, partitioning of the in-memory graph is done at the group-level of the fabric, where each node is designated data belonging to a proportionate range of groups in the fabric. In case of systems with uneven distribution of groups, switches, and ports, users can choose to configure their own distribution of Caddy based on their metadata.

B. Memory-Efficient Online Aggregates

Caddy achieves efficiency and scalability of the Bins' contents through dynamic merging and updates of aggregate statistics in the Bins using Welford's algorithm which offers a computationally efficient (single-pass) approach for incrementally updating the aggregate statistics as new data arrives. This method enables real-time updates and analysis without the need to recompute the aggregates over the entire data points, significantly reducing both computational complexity and memory footprint. While the core algorithm maintains only three variables ($mean$, m_2, and n) for mean and variance calculations, it can be extended to support higher-order moments like skewness and kurtosis (as shown in Algorithm 1.

Additionally, Welford's algorithm can be extended to allow for a set of Bins to be merged to create a Bin (of the same memory-footprint) that represents their cumulative domain. Welford's algorithm is particularly beneficial in our use-case where maintenance of real-time telemetry data can quickly overwhelm the limited memory capacity of the RAM. It keeps track of the evolving aggregates, allowing for continuous updates as new data arrives.

Bins are the key to data compression and efficient data storage in Caddy. As shown in Algorithm 1, we can maintain aggregates upto the 4^{th} order or moment by maintaining and updating simply 5 variables in-memory. This shows that for a set of supported telemetry counters, each Bin, irrespective of its temporal and fabric domain, has a constant memory footprint, keeping our in-memory storage at a stable value despite variations in volume of incoming telemetry. Additionally, by leveraging online updates, our data model can efficiently accommodate data updates over evolving input streams without compromising analytical accuracy, since Welford's algorithm has been demonstrably proven to be accurate for calculating aggregate statistics. This gives Caddy a robust and scalable solution for data management and analysis.

Utilizing Bins, coupled with Welford online statistics, allows us to achieve two key functionalities: (1) rapidly identifying subregions (Bins) that require creation or modification based on incoming data, and (2) perform efficient, decentralized updates across our computing cluster - overlapping partitions can easily be merged during query evaluations. Merging of Bins may be required either if - (a) a higher-level (lower-resolution) Bin is required to be generated from the contents of lower level Bins in case of dynamic generation of a Bin hierarchy (explained below) , or (b) in case of a distributed setting where partitioned Caddy Bins have overlapping bounds - in such scenarios, in certain corner-cases, aggregate results with overlapping Bin, identified by their matching metadata, would need to be merged to get accurate results.

Each Bin contains 2 main information: (a) telemetry and temporal metadata, used for identification and validation during telemetry queries, and (b) Welford's summary statistics, which represents the aggregate data queried for diagnostics over the domain represented by the Bin. These summary statistics for matching Bins is the main information returned to a client program in response to an analytical query. The metadata information also helps the Bins be aware of their immediate position in the hierarchical graph structure, as explained later.

Algorithm 1 Welford's Online Algorithm: Online updates to aggregates for every new entry

Initialize:
m1 $\leftarrow$ 0 (mean)
m2 $\leftarrow$ 0 (variance)
m3 $\leftarrow$ 0 (skewness)
m4 $\leftarrow$ 0 (kurtosis)
n $\leftarrow$ 0 (number of samples seen)
for each new data point x **do**
  $n \leftarrow n + 1$
  $\delta \leftarrow x - m1$
  $m1 \leftarrow m1 + \delta/n$
  $m2 \leftarrow m2 + \delta * (x - m1)$
  $\delta 2 \leftarrow \delta * \delta$
  $m3 \leftarrow m3 + \delta 2 * (x - m1) - 3 * m2 * \delta/n$
  $\delta 3 \leftarrow \delta 2 * \delta$
  $m4 \leftarrow m4 + \delta 3 * (x - m1) - 6 * m2 * \delta 2/n + 4 * m3 * \delta/n^2$
end for

C. Hierarchical Bins

We design the structure of Caddy to make it condusive to interactive analytical queries. To facilitate this, we organize it as a hierarchical organization of Bins (Fig. 3), similar to the concept of hierarchical data tiles [2], in descending order of their resolution, i.e., coarsest resolution at the topmost level. Bins are logically organized as a *multi-relational property graph* with each level grouping Bins of a particular telemetry and temporal resolution. The hierarchy is constructed using Bin metadata in the following order $time\_range$, $group\_id$, $switch\_id$, $port\_range\}$, as shown in Fig. 3. This hierarchical organization of bins based on the ordering of Bins' key metadata is configurable. This simple mapping between the Bins' granularity and the level of the hierarchical graph makes it easy to rapidly identify the set of Bins required to compute the results of a query and then filter from them based on query parameters.

Although Caddy is logically designed as a tree, we physically organize it as a set of hashmaps with each Bin maintaining sufficient metadata to identify its neighborhood. We circumvent the need to maintain actual links between Bins and its neighborhood and with the Bins that lie above or below it in the hierarchical layout of Caddy (parent and child Bins, respectively), which could compund the overall size of the in-memory framework in the case of a large interconnect fabric. We acomplish this by designing and maintaining each Bin's metadata such that it can be easily used to identify the metadata of any Bin in its neighborhood. Unlike traditional tree-based systems, Caddy avoids costly traversals leveraging this metadata information in each Bin for discovery of its immediate neighborhood in the logical graph which further reduces its memory requirement. Each Bin holds the entire key range info and can easily identify ($O(1)$ time) its immediate neighbors/parents/children in the telemetry/temporal space. For instance, we can easily summarize, from Fig. 3, that the parent of a Bin represented by the tuple $\{2023-12-21, G_8, S_{15}, P99\}$ is $\{2023-12-21, G_8, S_{15}, \_\}$.

Due to this easy inference of Bins and their hierarchical relationships, Caddy allows for conditional persistence of low-resolution (higher-granularity) Bins, i.e., except for the leaf-level Bins, none of the higher-level Bins need to be constructed during data ingestion, minimizing the overhead to ingestion throughput of the framework. This is because, in case of limited memory capacity of the host nodes, simply maintaining lower-granularity Bins allows for the dynamic computation of higher-granularity Bins on the fly during query evaluations since lower level Bins can be quickly merged to contruct higher-level ones. It is to be noted that merging of the Bins leverages Welford's algorithm, which unlike traditional calculations, and avoids expensive re-computation of the mean and variance from scratch, leading to significantly faster updates. This rapid merging allows Caddy to be optionally configured to selectively decide whether to persist lower-resolution (higher-level) Bins in-memory, leading to greater flexibility in terms of usage based on the fabric over which it is being used. Additionally, during a cold start, Bins can be retroactively constructed using historical data from a persistent storage (TimescaleDB in case of FAIO) and then populated/updated with live data and purged periodically based on staleness to maintain the last N hours of most recent telemetry aggregates in-memory.

D. Data Ingestion and Population over Caddy

As mentioned before, FAIO's live mode works in a distributed fashion with each node handling a portion of the overall telemetry data aggregation, partitioned by its group ID. Each node has its own partitioned Caddy graph handing data for its share of groups to ensure data locality. To keep the telemetry ingestion process seamless, leaf-level Bins are generated/updated during data ingestion, whereas the non-leaf Bins may be evaluated lazily during query evaluations. As mentioned above, we can selectively determine which level of non-leaf Bins get constructed during data ingestion and which get constructed on-the-fly during query evaluation.

Additionally, to ensure accuracy of results and avoid redundant reevaluation of non-leaf Bins, Caddy maintains a hierarchical **bitmap** that represents the consistency of Bins in that level. Since identification of a leaf-level bin's hierarchy is a simple operation, during ingestion/update, we go up the bitmap hierarchy to mark the Bins that have become stale in anticipation for future query. Ideally, due to the time-series nature of telemetry data, only a negligible section of a non-leaf Bins will become stale over time. It is to be noted that in cases where the full hierarchy of Bins is updated during data ingestion, there is no need to maintain the bitmap, since all Bins in Caddy would be up-to-date on ingestion.

E. Query Evaluation

Fig. 4 demonstrates the general structure of a telemetry query to be evaluated by the Fabric AIOps back-end against a visualization request from its front-end UI. The $\mathbf{R_f}$ and $\mathbf{R_t}$

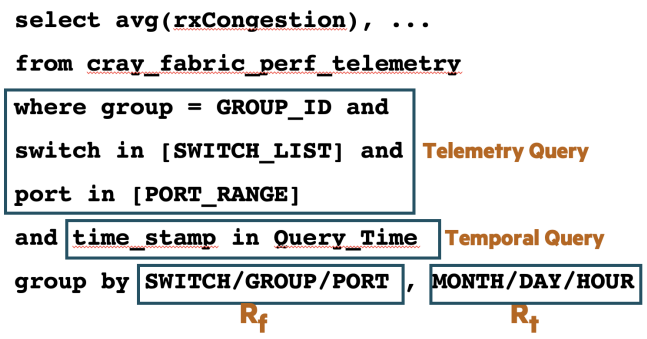

Fig. 4: Structure of a FAIO telemetry aggregation query over a fabric sub-sectiion

query parameters represent the fabric and temporal resolution at which telemetry is requested by the user, respectively. $\mathbf{TelemetryQuery}$ and $\mathbf{TemporalQuery}$ represent query filters that allow users to specify time ranges, as well as the cross-section of the fabric over which users' request telemetry aggregates.

During evaluation, $\mathbf{R_f}$ and $\mathbf{R_t}$ are used to identify the Caddy level from which Bins are to be queried - this helps us rapidly isolate the subset of Bins for each analytical evaluation. In cases of selective population of Caddy levels, we would look at the lower levels for candidate Bins to be used to construct the requested higher level (lower granularity) Bins. In the next step, once the relevant Caddy level is identified, the $TelemetryQuery$ and $TemporalQuery$ are used to filter Bins with metadata that match the query. Finally, Bins from individual Ray actors (partitioned Caddy datastores) are combined into a set of frames based on their timestamp to construct the response to the analytical query. Optionally, in case of overlap of Bins' metadata between actors, they can be rapidly merged using Welford's algorithm, as detailed previously.

F. Data Purging

In order to facilitate analytics over the most recent telemetry in live mode for real-time analysis, Caddy prioritizes maintaining the most recent Bins in case of a memory overflow. This overflow can be configured as a preset limit to either the total number of Bins that can be maintained in Caddy or as the difference between the current timestamp to the timestamp of its oldest entry. Crossing of this overflow threshold would trigger a purging of the in-memory Bins. Cady employs a targeted eviction scheme for its Bins that strategically removes older Bins to make space for newer ones containing the latest information to enable a sliding temporal window of in-memory Bins.

The eviction utilizes the timestamp metadata associated with each Bin to evaluate its relevance. Bins are purged based on ascending order of their timestamp metadata, where the topmost entries in the order would contain the Bins oldest data within the in-memory storage. Additionally, once stale Bins are identified at the temporal level, Caddy strategically purges all its children Bins in a top-down fashion, leveraging the easy identification of the subtree for each parent Bin purged by matching for the prefix of their metadata. This ensures that only is the oldest data removed within a timeframe, freeing up valuable in-memory storage for the latest information. This targeted eviction approach allows Caddy to maintain a fresh and up-to-date in-memory representation of the data, crucial for real-time analytics and insights, without much overhead to its query analytics.

It is to be noted that this maintenance scheme is executed parallely without halting ingestion or query evaluation over the framework. Also, purging of stale bins in Caddy is executed intermittently at configureble intervals.

V. System Evaluation

We benchmark our system based on telemetry data modeled after telemetry from HPE's Hotlum system. Hotlum is a high-performance computing (HPC) system built on HPE Cray EX hardware with 1024 nodes, each containing dual AMD EPYC 7763 processors and either 512 Gb or 1024 Gb of memory. The system utilizes HPE Performance Cluster Manager 1.10 for cluster management, COS 3.0 (based on SLES 15 SP5) as the operating system, Slurm 23.02.6 as the workload manager, and high-speed networking provided by Slingshot 11 with software version 2.1.1. The following evaluations were performed on an instance of FAIO deployed over a single node of a a high-performance computing system equipped with Gen 10+ 128 AMD EPYC 7002 Series (XL225) processors with 196 GB of memory. The fabric consists of 8 groups, each containing 8 switches, which in-turn contain 64 ports each. Overall, for our benchmarks, our system ingested continuous telemetry at the rate of 300K telemetry events recorded per minute, which occured parallely to any analytical queries.

A. Caddy Telemetry Data Compression Evaluation

As mentioned previously, data compression in Caddy is achieved through aggregating telemetry data into fixed-size Bins. The temporal size of these Bins is configurable and determined based on the underlying system and the users' requirements on how fine-grained a view of the system would suffice for the FAIO live mode. Here, we evaluate the compression ratio achieved for different Bin sizes. As expected, larger Bin sizes result in higher compression ratios due to the ability to represent higher volume telemetry within a single Bin of constant size and exploit temporal redundancy in the data. We evaluated on three Bin sizes: 10 minutes, 15 minutes, and 30 minutes and compression ratio increases linearly with Bin size, demonstrating the effectiveness of Caddy in reducing data storage requirements. This demonstrates that through aggregation of continously streaming data into fixed-sized Bins, we can significantly boost the storage capacity of the live mode in terms temporal range

B. Comparison of Fetch Times for Single-frame Snapshot

We also benchmark the improvement in latency of evaluations of analytical queries over Caddy compared to

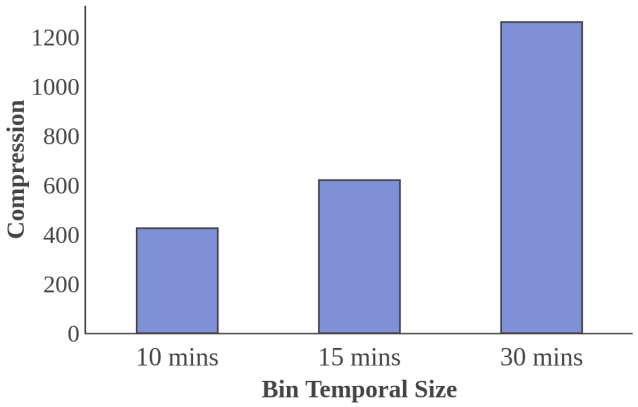

Fig. 5: Compression Factor vs Bin Size

TABLE I: Comparison of fetch times between live Mode and Caddy-enabled live mode

Single Frame Fetch Time		
Mode	**Time (ms)**	
Live Mode	355	
Live Mode + Caddy	Group Level	151
	Port Level	267.51

FAIO's live mode over single snapshots of the fabric. Table I shows a comparison between Caddy and traditional live mode for average evaluation times for aggregate queries over telemetry for the temporal window of 10 mins and varying fabric resolution levels. Traditional live mode currently does not support fabric-level aggregation - the 355 ms is the time taken for fetching port-level aggregate statistics. With Caddy, we demonstrate aggregateions at both group and port-level (i.e. we aggregate the value of the counter `rxCongestion` over each group/port). Due to Caddy's ability to identify in-memory data segments faster due to more efficient hierarchical indexing, we can see improved query latency. Additionally, it is to be noted that Caddy supports fetches over multiple time-segments over a larger requested time-range in the form of an array of snapshots of the fabric in a single query. So the improvement in latency between the FAIO live mode and Caddy is further amplified over larger time ranges. We demonstrate the latencies of such queries below.

C. Large-scale Telemetry Query Analytics

To evaluate the performance of Caddy, we profiled our system on a set of queries involving long temporal windows for telemetry data. These queries simulate real-world scenarios where users would request aggregated data over various timeframes grouped into 10-minute intervals (which is the highest-granularity for temporal Bins in our benchmarks). The queries retrieve and aggregate this data for a user-specified temporal window, and the results are presented as frames within a carousel interface, as shown in Fig. 6.

TABLE II: Comparison of data ingestion rates for FAIO with and without Caddy

	Time (s)
Live Mode + Caddy	1.007
Live Mode	0.7589

Fig. 6 illustrates the change in query latency with increasing temporal scope of the query. As expected, it shows a direct correlation between the length of the queried time window and the latency experienced. Queries encompassing longer durations requires look-up into greater number of potential candidate Bins and subsequent merging of selected Bins. Based on these findings, we can infer that for optimal user experience, the ideal temporal window size should not exceed 3 hours. For queries exceeding this timeframe, a pagination mechanism should be implemented, where queries are executed when sections of the overall carousel view are fetched based on the users' actions to enhance interactivity. This would allow users to navigate through the results in smaller, more manageable chunks, reducing perceived latency and improving overall responsiveness.

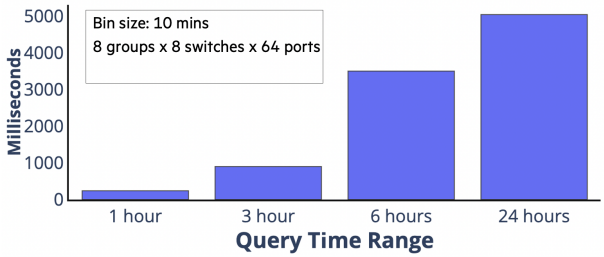

Fig. 6: Query Times vs Query Size

D. Ingestion Overhead

We profile the ingestion overhead of maintaining an in-memory tree of aggregates, compared to our current live mode strategy of maintaining recent telemetry in a arrays one for teach telemetry measurement (counters). Table II demonstrates the comparison of average ingestion time of FAIO's live mode with and without the Caddy in-memory framework. In this experiment, we compare compare the latency of ingestion of 1 million telemetry data randomly generated for a time-period representing 1 hour. We can see that population of in-memory data with Caddy is slower compared to that of our current live mode (32.5%). Given the improvements demonstrated in the previous evaluations, we propose that is an acceptable overhead for our in-memory store.

VI. Conclusion

Here we described, Caddy, our framework for interactive aggregations over voluminous telemetry datasets generated

from exascale fabrics.

RQ-1: Caddy facilitates interactivity for large-scale queries over the fabric. To achieve this, incoming high-velocity telemetry are consolidated into Bins over configurable temporal and telemetry ranges, leading to significant reduction of the memory-footprint of the stored data. Our scheme maintains enough information to generate aggregates upto the 4^{th} order of moment and avoids redundant computations during data ingestion and enables reusablility of results.

RQ-2: Our compact Bins are supplemented with metadata relating to their domain, enabling us to identify or merge them data for generating interactive visualizations over varying granularities. This supports efficient summarizations and hierarchical aggregations of telemetry data across different fabric or temporal scopes. We ensure accuracy of these aggregate metrics by utiliaing Welford's online algorithm, which allows us to maintain accurate representations of the underlying data in-memory at a constant memory footprint per Bin.

RQ-3: Stale Bins are purged based on their temporal metadata. We leverage the hierarchical nature of our data model to rapidly identify Bins with stale metadata. Starting at the temporal level, Bins can be sorted on their timestamps and identified for purge based on whether their timestamps are beyond the allowable threshold. We then use these shortlisted Bins at the temporal level to easily identify their sub-trees for elimination from Caddy to keep it up-to-date.

RQ-4: Maintaining summarizations at varying granularities helps us minimize duplicate processing over Caddy. This hierarchical structure facilitates easily identification of the set of Bins relevant to a particular query based on the requested resolutions and topology bounds. Additionally, we utilize the metadata of each Bin to precisely locate its relative position in the overall in-memory graph, allowing for rapid identification or merging across tree levels during the time of query evaluation.

VII. Future Work

We aim to further explore the scalability of Caddy in larger, more diverse HPC settings. This includes testing its performance and efficiency in exascale environments with higher data volumes and varying computational loads. Additionally, we plan to explore how integrating Caddy with existing or emerging technologies in HPC ecosystems which can create new opportunities for real-time data processing and analysis, for instance, using the compact data Bins for predictive models and checking for accuracy of results. These explorations could significantly broaden Caddy's utility and influence within the field.

References

[1] Welford, B. P. "Note on a method for calculating corrected sums of squares and products." Technometrics 4.3 (1962): 419-420.

[2] Lins, Lauro, James T. Klosowski, and Carlos Scheidegger. "Nanocubes for real-time exploration of spatiotemporal datasets." IEEE Transactions on Visualization and Computer Graphics 19.12 (2013): 2456-2465.

[3] Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S. and Stoica, I., 2010. Spark: Cluster computing with working sets. In 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10).

[4] Mitra, S., Khandelwal, P., Pallickara, S. and Pallickara, S.L., 2019, September. Stash: Fast hierarchical aggregation queries for effective visual spatiotemporal explorations. In 2019 IEEE International Conference on Cluster Computing (CLUSTER) (pp. 1-11). IEEE.

[5] Liu, Z., Jiang, B. and Heer, J., 2013, June. imMens: Real-time visual querying of big data. In Computer graphics forum (Vol. 32, No. 3pt4, pp. 421-430). Oxford, UK: Blackwell Publishing Ltd.

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

[7] Pahins, C.A., Stephens, S.A., Scheidegger, C. and Comba, J.L., 2016. Hashedcubes: Simple, low memory, real-time visual exploration of big data. IEEE transactions on visualization and computer graphics, 23(1), pp.671-680.

[8] Tao, W., Liu, X., Wang, Y., Battle, L., Demiralp, Ç., Chang, R. and Stonebraker, M., 2019, June. Kyrix: Interactive pan/zoom visualizations at scale. In Computer Graphics Forum (Vol. 38, No. 3, pp. 529-540).

[9] Batt, S., Grealis, T., Harmon, O. and Tomolonis, P., 2020. Learning Tableau: A data visualization tool. The Journal of Economic Education, 51(3-4), pp.317-328.

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

[11] Jain, A., Gupta, A., Gupta, A., Gedia, D., Pérez, L., Perigo, L., Gandotra, R. and Murthy, S., 2019. Trend-based networking driven by big data telemetry for SDN and traditional networks. arXiv preprint arXiv:1904.10449

[12] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M.J., Shenker, S. and Stoica, I., 2012. Resilient distributed datasets: A Fault-Tolerant abstraction for In-Memory cluster computing. In 9th USENIX symposium on networked systems design and implementation (NSDI 12) (pp. 15-28).

[13] Battle, L., Chang, R. and Stonebraker, M., 2016, June. Dynamic prefetching of data tiles for interactive visualization. In Proceedings of the 2016 International Conference on Management of Data (pp. 1363-1375).

[14] Srinivasan, M., Mallick, D., Maschhoff, K. and Ayyalasomayajula, H., trellis—An Analytics Framework for Understanding Slingshot Performance.

[15] Roweth, D., Faanes, G., Treger, J. and Terpstra, M., HPE Slingshot Launched into Network Space.

[16] Nielsen, J., 2009. Powers of 10: Time scales in user experience. Retrieved January, 5, p.2015.

[17] Karau, H. and Lublinsky, B., 2022. Scaling Python with Ray. " O'Reilly Media, Inc.".

[18] Welford, B.P., 1962. Note on a method for calculating corrected sums of squares and products. Technometrics, 4(3), pp.419-420.

[19] Weidner, O., Barker, A. and Atkinson, M., 2017, June. Seastar: a comprehensive framework for telemetry data in HPC environments. In Proceedings of the 7th International Workshop on Runtime and Operating Systems for Supercomputers ROSS 2017 (pp. 1-8).

[20] Jha, S., Cui, S., Banerjee, S.S., Xu, T., Enos, J., Showerman, M., Kalbarczyk, Z.T. and Iyer, R.K., 2020, November. Live forensics for HPC systems: A case study on distributed storage systems. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-16). IEEE.

[21] Timescale. (2024). Timescale Docs. https://docs.timescale.com/ [Accessed: 2024-04-04]

[22] Omnisci. (2024). OmniSci Overview. https://docs.heavy.ai [Accessed: 2024-04-04]