STREAM: A Scalable Federated HPC Telemetry Platform

Ryan Adamson, Tim Osborne, Corwin Lester, Rachel Palumbo
National Center for Computational Sciences
Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA.

***Abstract*—**
Obtaining and analyzing high performance computing (HPC) telemetry in real time is a complex task that can impact algorithmic performance, operating costs, and ultimately scientific outcomes. If your organization operates multiple HPC systems, filesystems, and clusters, telemetry streams can be synthesized in order to ease operational and analytics burden. In order to collect this telemetry, the Oak Ridge Leadership Computing Facility (OLCF) has deployed STREAM (Streaming Telemetry for Resource Events, Analytics, and Monitoring), which is a distributed and high-performance message bus based on Apache Kafka. STREAM collects center-wide performance information and must interface with many sources, including five HPE deployed supercomputers, each with their own Kafka cluster which is managed by HPCM. OLCF Supercomputers and their attached scratch filesystems currently send more than 300 million messages to over 200 topics producing around 1.3 Terabytes per day of telemetry data to STREAM. This paper describes the architectural principles that enable STREAM to be both resilient and highly performant while supporting multiple upstream Kafka clusters and other data sources. It also discusses the design challenges and decisions faced in adapting our existing system-monitoring infrastructure to support the first Exascale computing platform.

***Index Terms*—Observability, HPC Telemetry, Analytics, STREAM**

I. Introduction

The Oak Ridge Leadership Computing Facility (OLCF) [1] deploys and operates leadership-class computational resources required to tackle global challenges. OLCF deployed the CrayEX based Frontier supercomputer in 2022 which ranked fastest in the world by the Top500 list [2] and is the first computer to reach exascale at over 1 EF of performance. Frontier's Lustre based scratch filesystem, Orion, contains over 650 PB of available storage. In addition to these 2 flagship systems, OLCF operates the 200 PF Summit supercomputer and a handful of other CrayEX based computers. Due to the complexity and high number of systems within this operational environment, it is critically important but also quite difficult to collect the appropriate logs, metrics, and telemetry information from various components of production systems. OLCF has deployed STREAM (Streaming Telemetry for Resource Events, Analytics, and Monitoring) over the past two years to centralize the collection of this system information in a performant way for operations staff, center leadership, and researchers.

The paper is organized as follows: First, contributions to literature are discussed in section 2. Section 3 explores the design requirements for a scalable federated message bus shared by the OLCF as well as the other strategic partnership programs operated by the National Center for Computational Sciences (NCCS). Section 4 provides a comprehensive overview of STREAM as it was implemented. Section 5 describes the lessons learned while taking STREAM from conception to deployment. Finally, section 6 discusses the future work that the OLCF has identified in order to fill remaining gaps in the STREAM deployment. Finally, section 7 concludes the paper.

II. Contributions

This paper makes contributions in several areas. First, the diagrams and technical discussion contained within this text can be used as a starting point for highly-scalable federated streaming telemetry platforms that readers may wish to deploy. Second, several novel solutions to discovered federated streaming problems are discussed, such as federated schema registries, topic naming schemes, and consistent data dictionary formatting. Finally, because STREAM is a system that has evolved over the past few years and will continue to evolve as OLCF's needs grow over time, this paper serves as an experience report and lessons learned document. As such, care should be taken when adapting blueprints and designs to make sure they meet the needs of the implementer. Kafka is a complex message bus, it is potentially the wrong choice for simpler and/or less rigorous publish-subscribe workloads.

III. Design Requirements

Working with telemetry data from multiple supercomputers requires coordination, optimization, and planning. The volume of data a supercomputer produces must flow freely and avoid bottlenecks; as streaming continues to gain traction and OLCF adds more telemetry metrics and/or systems, the Analytics and Monitoring (AM) workload must be minimized and needs to scale linearly in order to continue providing leadership level streaming capabilities. The following design goals were present during deployment of STREAM:

1) Lower the burden of data engineering and pipeline management
2) Scale over time with the deployment and decommissioning of systems
3) Serve as a 'narrow waist' for reliable data flow within NCCS
4) Provide a data-agnostic abstraction layer for producers and consumers

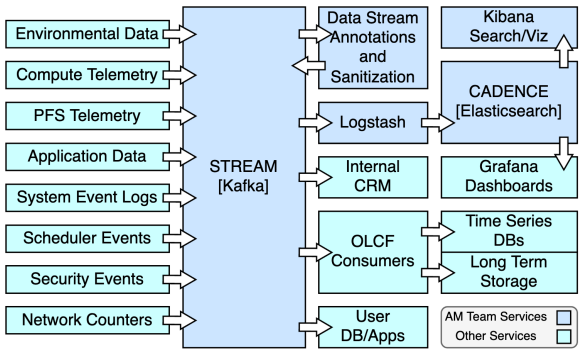

Fig. 1. STREAM is the narrow waist of OLCF's telemetry platform.

A. Minimizing Data Engineering Burden

In recent years, several papers have been published that principally used information about OLCF supercomputers to make discoveries. This work tends to involve reliability analysis, failure prediction, or characterization of efficiency of world-class resources. [3][4] Generally, the monitoring platforms present at sites serve in an operations role and system information is guarded by the systems engineers. There are several barriers to producing research outcomes in this model: Researchers must request (and be granted!) information from operations teams. Then, they must learn about and clean the data set they have received by performing exploratory data analysis (EDA). Finally, researchers must transform the data into a format that their analysis tools can operate on. This preparatory work can consume between 50% to 80% of the total effort spent! [5]. Additionally, future related research tends to *still* follow the same exact pathway and no shared-speedup is experienced even with related data sets.

To help minimize the mean-time-between-publications, the OLCF endeavors to perform as much of this curation work as is reasonable, and it is one of the guiding design goals of STREAM.

B. Data Centralization and Reliability

The additional design requirements of STREAM are operational in nature. We desire data access reliability, some guarantee as to the accuracy of information flowing through the platform, and a highly-scalable system capable of absorbing short bursts (such as during system failure events) as well as long-term sustained growth as more systems come online. The data pipelines that existed within the OLCF prior to STREAM were bespoke 'best-effort' connections between databases and scheduled cron tasks that were prone to silent failure. It was clear that the real issue standing in the way of reliable data streaming was not the systems themselves, but the number of point-to-point flows of information that existed without strong documentation and awareness. The OLCF wanted to centralize these in some way to provide operational and correctness guarantees to data pipelines.

Many complex systems take on an 'hourglass shape' [6] to minimize complexity between the many supporting layers of functionality. For example, the narrow waist of the internet is the IP layer. IP routing is relatively simple and is only concerned with distributing packets somewhat efficiently along a path between communication endpoints. Because the routing layer does not need to care about the supporting or supported layers of the OSI model, it has allowed global communication networks to grow significantly over the past forty years. STREAM is the 'narrow waist' of the OLCF's streaming telemetry data platform: Telemetry data flows from source to STREAM and then to the many consumers that are interested in subscribing to those streams.

C. Scaling Over Time

Another benefit of a centralized narrow-waist design over a direct all-to-all topography is that the worst-case number of connections between data sources and sinks is reduced from $O(n^2)$ to a $O(n)$ upper bound the n-to-1-to-n hourglass architecture. There are pros and cons to this approach, however. Centralization means that we can develop better monitoring tools and documentation for end users, but it can potentially pose a risk to overall throughput and scalability by becoming a bottleneck.

D. Lifecycle Sustainability

The sustainability of center-wide services is very important. Clearly, there is systems engineering and administration expertise within the OLCF, but this data curation effort needed to move up the stack a little bit to also focus on data exploration and interpretation for STREAM customers. The Analytics and Monitoring (AM) team decided to operate STREAM on top of OLCF's SLATE Kubernetes/OpenShift infrastructure so that operating system and systems administration expertise of its members could be replaced by data analysis skill sets, while retaining the application administration and troubleshooting knowledge that is required of an operational team. The SLATE layer of abstraction makes it easier to bring in new hardware for growth and hardware replacement purposes as well as providing a consistent set of configuration for test environments that can be spun up on the fly. There is a learning curve regarding kubernetes operations, however, and it was a calculated risk at the time to embrace that kind of PaaS which was not supported as well as today by vendors of the applications that we run.

E. Performance Requirements

Performance requirements for a center-wide streaming telemetry platform are somewhat orthogonal to the standard HPC center mantra of low-latency and high-throughput communication. Because the primary purpose of STREAM is a real-time streaming telemetry platform, we expect most of the consumers to be reading either the latest or at least very recently written data. On message ingest, brokers will forward messages to subscribers that are online and polling. For subscribers that sleep and wake up periodically to monitor incoming messages, information will tend to still be in broker application memory or page cache. Thus, fast disk I/O is required only in workloads where clients want to read all of the data currently written to a topic. These use cases are better

suited to searching or analytics engines like Splunk, Elastic, or Spark. In those cases, other researchers and operations staff could run a telemetry gathering process to collect and downsample information into a format best suited for their use case. The main benefit to fast bulk disk I/O is mainly for recovery purposes during a broker failure. Re-replicating the topic partitions affected by a broker failure can cause TBs of recovery traffic from all systems and put load on network interfaces that can affect other topics.

The OLCF has a goal to store and curate as much telemetry data as possible for the lifetime of OLCF systems. Bandwidth requirements are thus driven more by storage system sizes and less by bandwidth needs of incoming telemetry. If we expect 20PB of telemetry data over a 5 year period, a relatively meager 140 MB/s steady state data stream is sufficient and is about the bandwidth provided by a single gigibit ethernet link.

IV. Architecture Overview

The OLCF explored several publish-subscribe systems such as MQTT, RabbitMQ and Apache Kafka. Eventually, Apache Kafka was chosen due to the combination of an extremely scalable architecture and the fact that topic creation and naming, user access control, and monitoring can be tightly controlled by administrators. The High Performance Cluster Manager (HPCM) software delivered with the CrayEX platform tracks many local system metrics internal to a system in a Kafka cluster, and there is an added advantage of toolset compatibility between the central STREAM telemetry platform and cluster-local HPCM instances.

A. STREAM Software Components

The core service of STREAM is ***Apache Kafka***, which is a highly-scalable, distributed, and reliable publish-subscribe system. Kafka is one of the largest member projects of the Apache Software Foundation and has a rich ecosystem of supporting tools and documentation. Data producers and consumers interact with the Kafka message bus ***topics***, and a set of Kafka ***brokers*** parallelize the topic into a set of *partitions* which is Kafka's unit of scale. Clients connect to the set of Kafka brokers to publish messages or subscribe to topics. Brokers manage communication with all clients and track the state of the Kafka cluster in order to direct clients to other brokers that are managing the topic/partitions of interest. Brokers are responsible for enforcing guarantees of reliability and redundancy as appropriate.

Other important properties of Kafka include:

1) Messages are both unique (with the *offset* being the key) and ordered for any given topic-partition
2) Messages are guaranteed to be delivered in-order to consumers for a given topic-partition
3) Kafka brokers play a pivotal role in client orchestration, both ensuring that client producers distribute writes across topic partitions evenly, and client consumers take appropriate slices of work and do not consume messages that other consumers within the same consumer group have seen already.
4) Low-latency is desired for real-time streaming, so page cache and in-memory processing is preferred
5) Telemetry streams range from consistent to bursty depending on producer and consumer behaviour

Zookeeper is a companion service of Kafka that stores cluster state information such as the current leader, topic settings, and access control lists.

The **Schema Registry** is a service that stores message schemas for topics. Schemas can be used by brokers to enforce message consistency and enable validation. Schemas typically contain a list of fields, their data types, and comments describing each field. Topic schemas are important for **AVRO** encoding as well, which can drastically reduce data and processing demands using compression. If AVRO encoding is used while producing messages to a topic, clients will need to reference the Schema Registry to be able to decode topic messages.

Kafka Connect is a way to connect Kafka with remote data sources. It is a suite of plugins to ingest from and export data to other applications such as Elastic, Telegraf, and even other Kafka clusters. STREAM uses Kafka Connect to ingest data from the HPCM-local Kafka clusters within NCCS.

The **Kafka REST Proxy** service lets producers or consumers without native Kafka libraries speak using the Kafka message protocol. These clients produce or consume from a RESTful API interface for Kafka provided by the service.

Confluent Control Center provides monitoring and configuration capabilities for a Kafka cluster. Topics are provided dashboards that show data speeds and feeds as well as individual recent messages. This provides the AM team observability to verify the health and status of topics as well as debug producers messages as needed.

Stream Schema Relay is an OLCF-developed Flask application that disambiguates schema conflicts across HPCM clusters that are federated with STREAM. Since each HPCM system has its own Schema Registry, the Stream Schema Relay forwards requests from STREAM's Schema Registry service to the host systems' Schema Registry, allowing clients to retrieve the correct schema and decode their message.

B. STREAM Hardware Components

STREAM is composed of six Dell PowerEdge R740s with 128 GB of RAM, 24x12 TB SSDs, and 2 Intel Xeon Gold 32 core processors at 3.1 GHz. Each of these servers are joined to a RedHat OpenShift cluster. Confluent's Kafka Kubernetes operator is used to deploy six Kafka pods evenly across available hardware. Each pod is configured to use 16 threads (8 cores), 64 GB of memory, and 24 TB of disk. Pods for ephemeral services such as Zookeeper, Control Center, and Schema Registry are not particularly processor or memory-heavy and are co-scheduled on the same six OpenShift worker nodes.

C. STREAM Producers

Some notable STREAM producers include HPCM, Lustre system metrics obtained via telegraf, and xalt[7] job records. These producers are created and maintained by various groups

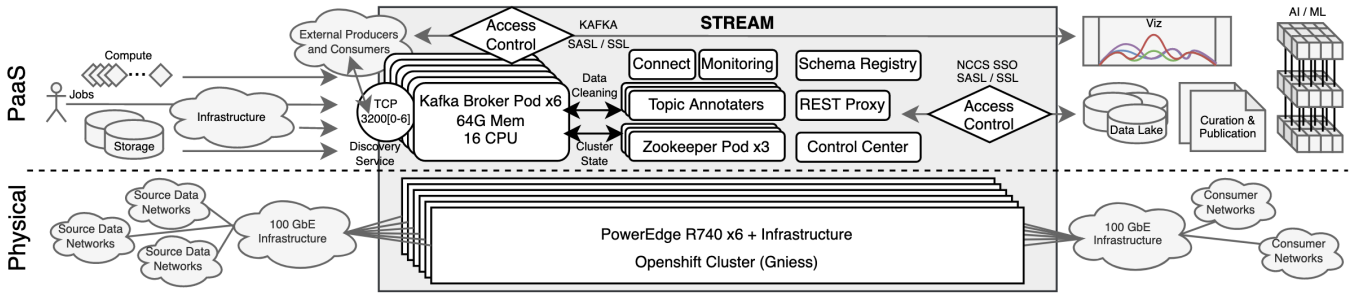

Fig. 2. Detailed STREAM components and their support by SLATE physical infrastructure. Data sources generally reside within their own network infrastructure on the left of the diagram while data sinks/destinations are on the right. SLATE also supports several data consumption applications.

throughout the OLCF. One of the heavier producers of data is HPE's HPCM cluster management software. HPCM produces through redfish to its own Kafka bus where the analytics and monitoring team replicates the stream using a Kafka replicator. Across NCCS systems, this averages around 18 MB/s at 105,000 messages/s. For Frontier data, the team augments some streams to include hostname and serial numbers for each component in the stream; we plan to replicate this enhancement to all the other equivalent HPCM streams. The storage team in OLCF collects and produces Lustre related data using telegraf. Storage accounts for about 50 MB/s of data and 128,000 messages/s on average. This data contains system telemetry for Lustre systems as well as rpc events from the systems. Another important but relatively low-profile producer to STREAM is the collection of xalt job records, which characterize applications compiled and run on HPC resources within the center. Of the 223 topics currently configured on STREAM, the top seven are detailed in Table I. The majority of data flows on STREAM are much smaller than these topics.

D. STREAM Consumers

Various consumers connect to STREAM with dashboards, exporters, or scrapers. One of the key applications that consumes data from STREAM is Summit's power efficiency application. This application uses real-time data from Summit and weather forecasts to optimize power for cooling. Most of the data that comes through STREAM is scraped and stored in an Elasticsearch Data Lake. Many Grafana dashboards across the OLCF are driven by the data in Elasticsearch. These dashboards provide operational awareness across the supporting organizations of OLCF.

E. Performance Analysis

Experience so far with STREAM has shown that the architecture is resilient and it has scaled well as new systems related to Frontier's delivery have been installed. Overall, STREAM's biggest pain point has been producers and consumers, not the Kafka bus itself. The current average write rate is 72 MB/s. The main producer being Frontier's file system, Orion, which bursts to 82 MB/s from 256 servers every few minutes. This pushes total throughput to 105 MB/s during that interval, with a sustained 38 MB/s when Orion isn't producing. We've observed 4 Kafka brokers sustain this write-rate, so we know we still have room to grow. Each Kafka node is outfitted with a 10 GbE connection for data. With six nodes, the theoretical maximum throughput is 7.5 GB/s, including replication, production, and consumption. Currently, we're at 300 MB/s including all replication, production, and consumption. So, the system can still grow 10x and be below half the theoretical max.

V. Lessons Learned

A. Topic Naming Conventions

Topic names should be informative but not cumbersome so that streams are differentiated in a standard and reliable way. This prevents mislabeling, confusion, and loss of data throughout the entire data life cycle. Topics are named according to the project, system, and type of data collected, such as metrics, telemetry, or power. We went through numerous iterations before landing on a set of naming conventions that covered both the information we wanted and constraints of our services requirements. Our current topic naming convention is a tuple of identifying information about a data source and takes the form:

DataOwner.SourceSystem.Subsystem.CommonName

As an example, the HPCM messages containing CrayEX telemetry information and other metrics for Frontier would be named:

stf002hpc.frontier.hpcm.crayex_telemetry

Here, *stf002hpc* is the name of the Unix group that corresponds to the systems administrators of HPC systems at NCCS. Note that this scheme makes finding crayex_telemetry information for any HPCM system within NCCS relatively simple since replacing 'frontier' with another system name would yield a valid topic name with that information in it.

NCCS updated our topic naming conventions simultaneously with hardware and architecture updates of STREAM and encountered a few data management issues. Streams were updated with new names based on the new conventions by creating new topics while retaining the same data pipeline architecture. However, in a few cases some streams were recreated with improper names and the data that was then stored at the endpoint (in Elasticsearch indices) had to be re-indexed into new, properly named indices. This can be a slow process, so the lesson here is to ensure you have a few fail safes in

Topic Name	Messages per second (avg)	Messages per second (max)	Bytes per second (avg)	Bytes per second (max)
orion.lustre.rpc_trace_data	111 K	246 K	43 MiB	85.9 MiB
frontier.hpcm.crayex_telemetry.	59.3 K	59.8 K	5.96 MiB	6.04 MiB
c5.hpcm.crayex_telemetry	10.2 K	10.9 K	1.05 MiB	10.7 MiB
frontier.hpcm.HPCMLOG	9.48 K	16.5 K	3.68 MiB	6.75 MiB
f2.lustre.rpc_trace_data	5.54 K	18.7 K	1.83 MiB	5.07 MiB
infrastructure.system.metrics	4.83 K	5.79 K	3.36 MiB	3.39 MiB
summit.syslog.messages	2.16 K	9.66 K	945 KiB	3.07 MiB

TABLE I
THE TOP SEVEN STREAM PRODUCERS AS OF MAY 2023. OF STREAM'S 220 TOPICS, THESE SEVEN DOMINATE BOTH MESSAGE COUNT AND DATA RATE OF ALL MESSAGES FLOWING THROUGH STREAM

place and have decided on a final naming convention before moving forward with any changes to production pipelines.

B. Federated schema registries

As a centralized Kafka bus which absorbs many Kafka buses, the various schema registries that match absorbed Kafka buses must be made available to consumers. These schema registries may be required in order to read the data, as is the case with encoded messages. We considered a few options for allowing consumers access to schema registries, the most obvious being consuming the schema registry messages from federated Kafka schema topics and storing them within STREAM's schema topic. Unfortunately, Kafka expects the central schema registry topic to exist before spokes. This is because encoded messages reference with registry index they're using. Rather than develop complex schema syncing and message modification tools, we decided to proxy user requests for topics that might have conflicting schemas between federated Kafka instances.

C. Access Approval Workflow

A data access and distribution workflow is absolutely necessary for maintaining data quality, integrity, confidentiality, and ensuring appropriate use. Misuse and manipulation of data, whether malicious or not, can lead to false, misleading, or negative conclusions that may be damaging and lead to costly mistakes and misunderstandings. Some of the difficulties faced in developing and implementing an access and distribution workflow lie in the disparate nature of the approval process: data producers, data requesters (internal to ORNL and external), security administrators, data service administrators (i.e., our team) and legal reviewers work on different teams, groups, or even facilities and have different requirements, considerations, and constraints on the data. In order to alleviate some of the strain of this process, we helped spearhead a Data Resource Utilization Council (DRUC) to manage and track data sharing and distribution. Within the DRUC, we developed a standardized workflow and Jira ticketing project to track and record progress of data sharing efforts.

D. Monitoring

The team monitors our Openshift pods and services using Prometheus exporters and Grafana dashboards. In additon, we have Nagios alerting set up to notify if we have significant broker failure. During downtimes or instances in which we experience signficant broker outages due to breaking changes or updates, we disabled alerting in order to reduce workload on the facilities operators that monitor the 24/7 Nagios alerting system.

Monitoring data streams *themselves* is vital for ensuring the integrity of the services STREAM offers. We currently monitor our topics using Prometheus and Confluent Control Center. Prometheus scrapes information such as broker health and status, message count per topic, production and consumption throughput, as well as production by broker. Control Center also offers this view in addition to a GUI for quickly viewing incoming messages.

E. Documentation of topics

Creating standardized documentation is difficult across datasets that are managed and produced by different groups. It is however, extremely important for adequate management and sharing of data. Our documentation process is ongoing. We decided to centralize our topic documentation in GitLab. While it is still in progress, we are working on implementing a CI/CD pipeline for managing and linting new topic documentation. Additionally, we have created standardized templates for topic documentation to allow for a more efficient and effective documentation process. Each of these things help to address a problem we have faced with incomplete, inconsistent, and confusing documentation that was produced without a standard template at different times or by different people.

F. Cleaning and purging unneeded data

Recent team efforts have been focused on analyzing and understanding incoming data in order to improve not only the efficiency and integrity of our services but to provide suggestions to data producers about how to better use and understand their data. One effort here is the identification of data fields that either provide no useful information for monitoring the system in question or contain null values. There is measurable physical storage and processing power needed for this data, so reducing the amount of resources needed to process and keep unnecessary data is vitally important.

G. Data wrangling

As with cleaning and purging data, wrangling data into a usable format is a nontrivial task. In some cases, this means identifying when one topic contains too much disparate information and needs to be separated into several different

topics or streams. Additionally, much of the data that we encounter is produced by first generation systems, therefore sensors, metrics, and telemetry data has not been standardized into a universal format and may as we have found, change throughout the lifetime of the data. This can cause significant and unexpected errors. Having standards for data set at the front end and good monitoring makes this aspect a little less disruptive.

VI. Future Work

A. Automating Topic Creation and Access

As the number of topics and consumers continues to grow, it has become increasingly important to develop mechanisms that enable users to create and access topics efficiently. OLCF has already implemented the Resource Allocation Tracking System (RATS) to facilitate the creation and management of projects and allocations for supercomputing resources. Plans are underway to extend RATS for managing streaming resources as well. Upon completion, users will be able to submit topic creation and access requests through MyOLCF, the web-based front end for RATS. The Analytics and Monitoring team can then approve these requests, automatically creating the required objects and permissions on the relevant services.

B. Training and Examples

With the ability to create and access streams independently, users must be educated on how to effectively interface and utilize services like Kafka and Elasticsearch. They will need to learn how to properly deploy new producers and consumers for streams. To assist users in this endeavor, we are working on providing well-documented examples for popular software libraries in multiple languages through MyOLCF. This approach will ensure that the scientific and engineering-focused audience can easily understand and apply the provided information to enhance their streaming capabilities.

C. Data Exploration Tools

As streams become publicly available, users need the ability to browse streams and examine data within them to uncover practical applications for the stream's data. MyOLCF will offer a data dictionary displaying meta-information about each stream, such as ownership, data fields present in each message, and the meaning of those fields.

In addition, we are exploring tools for analyzing data within a stream, such as incorporating a JupyterLab instance for running common data analytics libraries and statistics modules. Of particular importance are tools that enable users to visually create transformation and filtering pipelines, enhancing the value and utility of the data.

D. Broadening Scope

With a scalable, secure, streaming ecosystem capable of handling petabytes of storage alongside supercomputers, there is immense potential to expand the system's applications beyond the computing systems within OLCF. ORNL boasts powerful scientific instruments, such as the spallation neutron source and isotope production facilities, which can benefit from incorporating supercomputing into their processes. By providing adequate software tooling, pipelines can process streamed measurements, enabling experiments to run more effectively and facilitating the identification of anomalies or scientifically valuable data.

VII. Conclusion

STREAM is a scalable telemetry platform that centralizes the myriad data streams within NCCS and federates CrayEX HPCM Kafka message buses. It provides a center-wide view of real-time information for systems administrators, center leadership, and researchers that use system telemetry information for data science and machine learning. The NCCS has learned several important lessons during the deployment of STREAM which are layed out in this paper. In the future, we hope to move forward to democratize access to system telemetry information and provide streaming message bus as a service to OLCF customers.

VIII. Acknowledgements

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] Oak ridge leadership computing facility. https://www.olcf.ornl.gov/. [Online]. Available: https://www.olcf.ornl.gov/

[2] J. Dongarra and P. Luszczek, *TOP500*. Boston, MA: Springer US, 2011, pp. 2055–2057. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_157

[3] B. H. Park, S. Hukerikar, R. Adamson, and C. Engelmann, "Big data meets hpc log analytics: Scalable approach to understanding systems at extreme scale," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, 2017, pp. 758–765.

[4] W. Shin, V. Oles, A. M. Karimi, J. A. Ellis, and F. Wang, "Revealing power, energy and thermal dynamics of a 200pf pre-exascale supercomputer," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476188

[5] For data scientists, janitor work is key hurdle to insights. https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html. [Online]. Available: https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html

[6] M. Beck, "On the hourglass model," *Communications of the ACM*, vol. 62, 07 2016.

[7] K. Agrawal, M. R. Fahey, R. McLay, and D. James, "User environment tracking and problem detection with xalt," in *Proceedings of the First International Workshop on HPC User Support Tools*, ser. HUST '14. IEEE Press, 2014, p. 32–40. [Online]. Available: https://doi.org/10.1109/HUST.2014.6