Towards Unified Data Ingestion and Transfer for the Computing Continuum

Muhammad Arslan Tariq*, Ovidiu-Cristian Marcu*, Grégoire Danoy†*, Pascal Bouvry†*
*SnT, University of Luxembourg
†FSTM/DCS, University of Luxembourg
{arslan.tariq, ovidiu-cristian.marcu, gregoire.danoy, pascal.bouvry}@uni.lu

***Abstract*—The computing continuum can enable new, novel big data use cases across the edge-cloud-supercomputer spectrum. Fast and high-volume data movement workflows rely on state-of-the-art architectures built on top of stream ingestion and file transfer open-source tools. Unfortunately, users struggle when faced with dealing with such diverse architectures: stream ingestion was designed for small-size datasets and low latency, while file transfer was designed for large-size datasets and high throughput. In this paper, we propose to unify ingestion and transfer, while introducing architectural design principles and discussing future implementation challenges.**

***Index Terms*—Data ingestion, file transfer, unified architecture, fast data, computing continuum**

I. Introduction

Big data applications are becoming more complex due to larger datasets and the role of AI in accelerating scientific and industrial knowledge. This complexity necessitates the availability of high-performance computing (HPC) infrastructure. Consequently, there is a pressing need for research into new architectural approaches and optimizations that enable HPC and cloud centers to benefit from each other. Additionally, users require advanced support for efficient deployment across both cloud and HPC environments. This support should facilitate fast dataset orchestration between the two, optimizing performance and reducing costs at various application stages. Therefore, the computing continuum can enable new, novel big data use cases across the edge-cloud-supercomputer spectrum [1]–[3].

Several industries (e.g., finance, retail, smart cities, healthcare) can significantly benefit from integrating real-time data with historical data using a unified data ingestion and transfer architecture. This approach is vital across sectors for making informed decisions, improving efficiency, and innovating services. The research and industry communities are looking to derive knowledge and improve business outcomes faster by processing datasets at a rapid pace. To realize this urgency, users rely extensively on low-latency stream data ingestion and high-throughput file-based data transfer (as illustrated in Figure 1). However, it is difficult to handle such different, complex architectures, and no open-source fast data movement tool exists to efficiently reconcile both aspects.

Amazon's approach to data processing involves using distinct services for handling different types of data. For real-time data streaming, particularly for small records up to 1 MB, they recommend using Amazon Kinesis Data Streams. For larger datasets, Amazon proposes using Amazon S3, a storage service that can handle large amounts of data efficiently [4]. However, this approach, while effective in cloud environments, presents challenges when applied to HPC/edge computing scenarios, may not be easily or feasibly replicated in HPC or edge contexts due to their unique computational and data handling requirements and resource constraints.

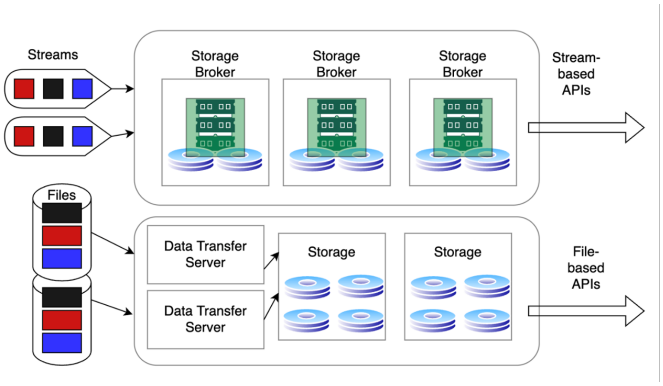

Fig. 1. To efficiently orchestrate both small stream record ingestion and large file data transfers, users have to manage two different deployments and APIs for stream ingestion (top) and file data transfer (bottom).

Our challenge is then **how to design and implement an optimized and unified architecture for data ingestion and file transfer** while keeping the advantages of fast data ingestion (e.g., low latency for small-size datasets, i.e., KBs to MBs) and high-volume data transfer (e.g., high-throughput for large datasets, GBs to TBs). Towards this goal and for efficiently optimizing data movement (low latency, high throughput, common storage layer, and unified read APIs), this paper aims to engage the research community and discuss our proposal: a novel architecture for efficient data movement that will efficiently and easily integrate new computing continuum use cases (e.g., such as those envisioned by sky computing [5]). We make the following contributions:

- Stream ingestion and file transfer traditionally respond to different data movement use cases and performance targets. To understand such divergent architectures, we choose to benchmark (two open-source relevant tools) Apache Kafka (for data ingestion) and Rucio (for file transfer) while considering small (KBs) to medium (MBs) dataset record sizes. Then, we comment on performance

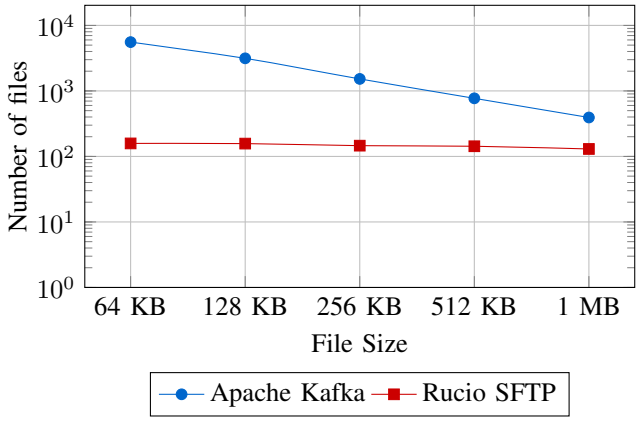

Fig. 2. Small file-size data write (KBs to MB). Measuring the number of files per minute.

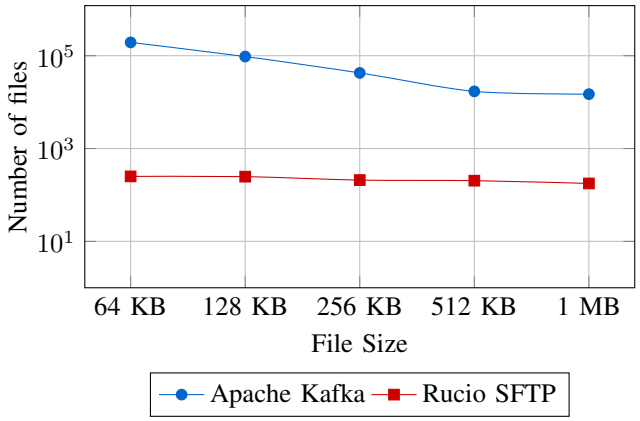

Fig. 3. Small file-size data read (KBs to MB). Measuring the number of files per minute.

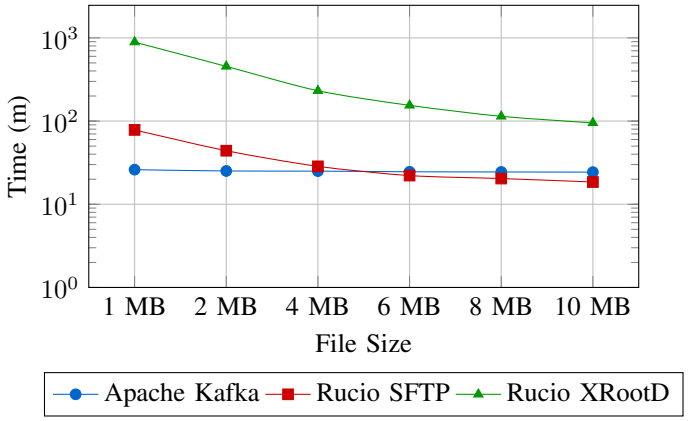

Fig. 4. Medium file-size data write (MBs). Runtime to ingest or transfer a 10GB dataset.

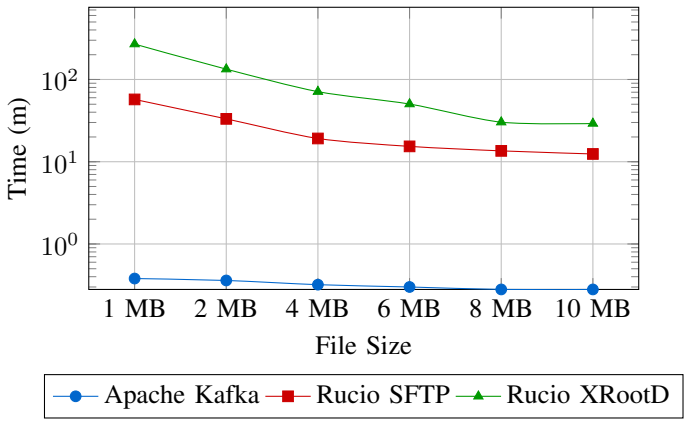

Fig. 5. Medium file-size data read (MBs). Runtime to ingest or transfer a 10GB dataset.

differences that motivate our unified architecture (Section II).

- We survey recent data ingestion and transfer tools and discuss their performance and API usage differences (Section III).
- We introduce and discuss a set of design principles to simplify a novel data movement architecture by minimizing storage copies and unifying read data access API (Section IV).
- We propose a novel unified ingestion and transfer architecture and discuss future implementation challenges (Section V), and then conclude (Section VI).

II. Background: Stream Ingestion versus File Transfer

To understand the performance (e.g., write and read throughput for various datasets) of stream ingestion and file transfer, we select two state-of-the-art tools. Apache Kafka [6] is a partitioned and replicated stream ingestion system providing low-latency access to small records (KBs to MBs). Rucio [7] is a file-based transfer solution used for scientific data, offering high-throughput access to large files (GBs to TBs). For example, machine learning workloads are built on transferred datasets with similar sizes, up to 1 TB [8]. We choose two configurations for Rucio: the first is based on the Secure File Transfer Protocol (SFTP), and the second leverages XRootD, providing POSIX-like access to files and their namespace directories.

Experimental setup and results. We configure the latest versions of Kafka and Rucio on a Linux server with 8 Intel Xeon processors and 128 GB RAM, deploying them in Docker containers. We implement write and read clients that transfer data from memory to memory, with the total dataset size ingested/transferred being 10 GB. Kafka is configured with one broker and one partition, while Rucio deploys one file transfer server. We measure the time taken to write and read a 10 GB dataset composed of files with various record sizes. As expected, Kafka outperforms Rucio by orders of magnitude when dealing with smaller record files (less than 1 MB) for both read (Figure 3) and write (Figure 2). Increasing the record size from 1 MB to 10 MB, we observe that Rucio gets more competitive with Kafka (Figures 4 and 5).

We also compared Rucio SFTP and Rucio XRootD across file sizes of 10-100 MB to evaluate their read/write throughput

performance. While Rucio SFTP focuses on secure file transfers, Rucio XRootD excels in handling large datasets. Our tests showed that Rucio XRootD outperformed Rucio SFTP in write operations for files larger than 90 MB and in read operations for files over 45 MB. These results suggest that Rucio XRootD offers superior throughput as file sizes increase.

These experiments contrasting stream ingestion (using Apache Kafka) with file transfer (using Rucio) underscore the need for a unified approach in managing diverse data requirements in big data applications.

III. Related Work

Ingestion systems [9] efficiently acquire, buffer, and temporarily store in-memory small record sizes and provide low latency and high throughput access to data streams. Confluent [10] is a commercial event-streaming platform that builds upon Apache Kafka's core functionalities for real-time data processing. It operates on a subscription model and offers several frameworks for data streaming, along with horizontal scalability through additional brokers. However, it has limitations in customization and is not suited for extensive file-based data transfers. Moreover, Apache Kafka relies on static stream partitioning and offset-based record access, trading performance for design simplicity. KerA [11], a data ingestion framework that alleviates these limitations, employs a dynamic partitioning scheme and lightweight indexing to improve throughput, latency, and scalability. Apache Pulsar [12] provides a cloud-based alternative to Kafka.

Rucio is a data management system designed for handling scientific data, particularly in distributed computing environments. It is often used in large-scale scientific experiments and collaborations to manage petabytes of data stored in billions of files distributed over 120 data centres globally. Rucio offers high-throughput data transfer and provides functionalities for data distribution, replication, monitoring, and metadata management. It is highly scalable and customizable, making it a preferred choice for complex scientific data workflows.

Skyplane [13] is designed for high-performance and bulk data transfers of large objects between inter-cloud object stores. The tool comprises two main components: a planner and a data plane. The planner formulates the job based on user requirements, while the data plane executes the transfer plan, handling data movement between clouds and interactions with object stores. The data plane also efficiently provisions virtual machines within the cloud infrastructure, all orchestrated via cloud-aware overlay networks. However, it's important to note that Skyplane does not support event-based streaming or local storage transfers.

Fivetran [14] offers a secure, SSL-encrypted connection to various supported data sources for applications, events, and data files, facilitating end-to-end data encryption. It uses data pull connectors to load data from multiple sources into a single destination. However, Fivetran's primary focus is on data replication, not complex data transformations, and it has limited customization options for unique data integration requirements. Multiple connectors can be set up to run as independent processes, each responsible for loading data from one or more supported data sources into one or multiple destinations. These connectors persist for the duration of each update cycle.

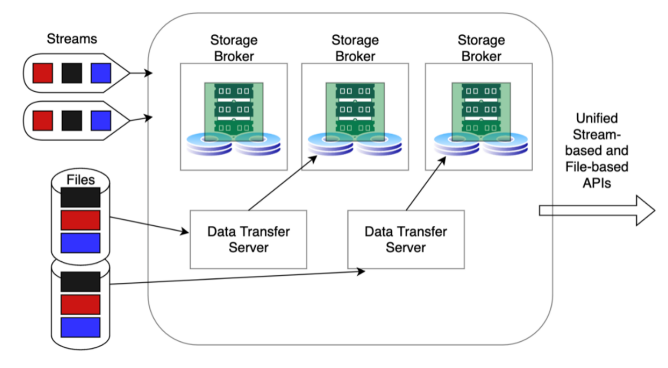

Fig. 6. Our proposal: leverage a common storage layer for a unified ingestion and transfer architecture for fast data movement.

SciStream [15] is a middlebox-based architecture designed to enable efficient and secure memory-to-memory data streaming in scientific applications, particularly when direct network connectivity is lacking. Using control protocols, SciStream establishes authenticated and transparent connections between data producers and consumers in HPC environments.

In summary, unifying data ingestion and transfer systems presents architectural challenges due to their distinct focus areas. Ingestion systems like Kafka excel in streaming and real-time data processing but are not designed for large file transfers. KerA attempts to mitigate some limitations with dynamic partitioning but remains focused on ingestion. Data transfer/movement tools do not support stream ingestion. No one-size-fits-all solution exists, making the unification a challenging endeavor.

IV. Fast Data Movement Design Principles

Remember that our goal is to design an efficient, fast data movement architecture that allows users to benefit from both stream ingestion and file transfer. This can be possible by relying on existing tools, e.g., Kafka/KerA for ingestion and Rucio for transfer. Therefore, users will leverage their standard open-source write APIs. However, data should be stored efficiently, whether small or large. Therefore, a unified approach should rely on a common storage layer shared by both ingestion and transfer services. Moreover, users should be able to access the acquired data with a unified read API. To simplify user access, a common metadata indexing layer (e.g., implemented by a key-value store) should also be considered as a first design principle. A unified, fast, and high-volume data movement architecture will rely on these two design principles while leveraging existing open-source ingestion and transfer tools. Our challenge is to realize a transparent integration of these tools while optimizing storage and simplifying user access.

V. Fast Data Movement Architecture and Implementation: Our Proposal

A. Proposed unified architecture.

Our goal is to design and implement a unified storage architecture integrating file-based transfer and stream ingestion to manage small to large record sizes. Our main challenge is *how to minimize data copies and improve users' productivity* for managing the movement of data through a unified ingestion and storage architecture based on open-source components. One approach is presented in Figure 1: we can leverage two separate deployments for handling stream ingestion and file transfer. However, forcing users to rely on two read APIs and writing small data through stream ingestion while writing large data through file transfer will also increase the complexity of the whole data movement workflow. Therefore, our second approach is a novel architecture that is presented in Figure 6: both transfer and ingestion will rely (as needed) on the common storage and indexing layers while presenting a unified read API to users. This new approach presents a set of challenges related to:

- The common storage layer requires both in-memory (streaming) and on-disk (file transfer) support. We have to decide at runtime which storage path to use.
- Read and write access consistency semantics have to be properly managed by the unified architecture considering application availability requirements [16].
- Reconcile aspects related to low latency and high throughput data access for such heterogeneous data sets.

Next design and implementation steps. We plan to use Apache Kafka as the frontend for cloud data access, complemented by KerA for in-memory dynamic ingestion. For implementation, we aim to enhance Skyplane's capabilities by integrating it with Apache Kafka to additionally support stream ingestion while creating an integrated cloud-HPC solution. To offer real-time metadata access for billions of files and records, we are considering a unified approach that combines in-memory storage KerA with a low-latency key-value store like RAMCloud [17]. In future work, we plan to scale our experimental setup by considering TB datasets with records in the KB-GB range.

VI. Conclusion

This abstract discusses the timely need to unify stream ingestion and file transfer in an optimized, open-source, fast data movement architecture. In particular, we evaluate Kafka and Rucio for small to medium dataset sizes and show their different read and write performance. We introduce and discuss the common storage layer and indexing principles and propose a novel unified architecture for stream ingestion and data transfer to minimize storage costs and simplify user experience through a unified read API. Finally, we identify future implementation challenges. We hope our proposal can pique the interest of the research and industry communities to help us further refine our assumptions and the architectural design proposal. Our architectural proposal, while maintaining existing write APIs (e.g., Kafka, Rucio) and implementing a unified read API, should benefit users by simplifying complex data workflows, optimizing infrastructure deployment, and reducing operational costs.

VII. Acknowledgment

This work is partially funded by the SnT-LuxProvide partnership on bridging clouds and supercomputers and by the Fonds National de la Recherche Luxembourg (FNR) POLLUX program under the SERENITY Project (ref.C22/IS/17395419).

References

[1] D. Park, "Future computing with iot and cloud computing," *the Journal of Supercomputing*, vol. 74, pp. 6401–6407, 2018.

[2] J. Y. Fernández-Rodríguez, J. A. Álvarez-García, J. A. Fisteus, M. R. Luaces, and V. C. Magaña, "Benchmarking real-time vehicle data streaming models for a smart city," *Information Systems*, vol. 72, pp. 62–76, 2017.

[3] M. Attaran and J. Woods, "Cloud computing technology: improving small business performance using the internet," *Journal of Small Business & Entrepreneurship*, vol. 31, no. 6, pp. 495–519, 2019.

[4] Masudur Rahaman Sayem, "Processing large records with Amazon Kinesis Data Streams," https://aws.amazon.com/blogs/big-data/processing-large-records-with-amazon-kinesis-data-streams/, 2023.

[5] S. Chasins, A. Cheung, N. Crooks, A. Ghodsi, K. Goldberg, J. E. Gonzalez, J. M. Hellerstein, M. I. Jordan, A. D. Joseph, M. W. Mahoney, A. Parameswaran, D. Patterson, R. A. Popa, K. Sen, S. Shenker, D. Song, and I. Stoica, "The sky above the clouds," 2022.

[6] K. M. M. Thein, "Apache kafka: Next generation distributed messaging system," *International Journal of Scientific Engineering and Technology Research*, vol. 3, no. 47, pp. 9478–9483, 2014.

[7] M. Barisits, T. Beermann, F. Berghaus, B. Bockelman, J. Bogado, D. Cameron, D. Christidis, D. Ciangottini, G. Dimitrov, M. Elsing *et al.*, "Rucio: Scientific data management," *Computing and Software for Big Science*, vol. 3, pp. 1–19, 2019.

[8] K. Park, K. Saur, D. Banda, R. Sen, M. Interlandi, and K. Karanasos, "End-to-end optimization of machine learning prediction queries," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 587–601.

[9] O.-C. Marcu, A. Costan, G. Antoniu, M. S. Pérez-Hernández, R. Tudoran, S. Bortoli, and B. Nicolae, "Storage and ingestion systems in support of stream processing: A survey," 2018.

[10] Confluent.io, "Apache Kafka vs Confluent," https://www.confluent.io/apache-kafka-vs-confluent/, 2023.

[11] O.-C. Marcu, A. Costan, G. Antoniu, M. Pérez-Hernández, B. Nicolae, R. Tudoran, and S. Bortoli, "Kera: Scalable data ingestion for stream processing," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1480–1485.

[12] K. Ramasamy, "Unifying messaging, queuing, streaming and light weight compute for online event processing," in *Proceedings of the 13th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 5. [Online]. Available: https://doi.org/10.1145/3328905.3338224

[13] P. Jain, S. Kumar, S. Wooders, S. G. Patil, J. E. Gonzalez, and I. Stoica, "Skyplane: Optimizing transfer cost and throughput using {Cloud-Aware} overlays," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1375–1389.

[14] Fivetran Inc., "Fivetran," https://www.fivetran.com/, 2023.

[15] J. Chung, W. Zacherek, A. Wisniewski, Z. Liu, T. Bicer, R. Kettimuthu, and I. Foster, "Scistream: Architecture and toolkit for data streaming between federated science instruments," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, 2022, pp. 185–198.

[16] E. A. Lee, R. Akella, S. Bateni, S. Lin, M. Lohstroh, and C. Menard, "Consistency vs. availability in distributed real-time systems," 2023.

[17] J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S. J. Park, H. Qin, M. Rosenblum, S. Rumble, R. Stutsman, and S. Yang, "The ramcloud storage system," *ACM Trans. Comput. Syst.*, vol. 33, no. 3, aug 2015. [Online]. Available: https://doi.org/10.1145/2806887