Toward a Push-based Stream Programming Model with AIMSS: An Active In-Memory Storage System Approach (Vision Paper)

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

Abstract—**Today's passive (on-disk and/or in-memory, employing a pull-based data access approach) storage architectures are performance- and energy- insufficient for handling the data-intensive demands of tomorrow's exascale machine learning and artificial intelligence (ML/AI) workloads. Industry projections forecast beyond-exascale clusters consuming energy between 500 MW and 1 TW, highlighting the need for a paradigm shift in data movement and processing, necessitating novel solutions that can improve performance, reduce energy consumption, and simplify application development and deployment. We believe exascale computing will require in-memory storage systems with a global perspective on I/O and processing, strategically positioned between traditional disk-based storage systems and CPU-GPU compute engines. We present the vision for an Active In-Memory Storage System (AIMSS), a novel architecture that shifts data movement management, such as source/sink handling and data shuffling, from ML/AI applications and big data streaming engines, directly to AIMSS. Operating on a log-structured in-memory storage framework, leveraging immutable data access patterns, and facilitating efficient real-time data movement, the AIMSS architecture will be deployed on tens of thousands of large many-core CPU-GPU nodes, harnessing their memory and ensuring efficient and transparent communication with traditional disk-based file storage systems. We propose a push-based streaming execution model enabling AIMSS to cost-effectively harness application-specific data (such as consumer/producer offsets and data access patterns including read, write, and shuffle) and thereby enable a set of optimizations such as scalable data movement partitioning algorithms, faster stream storage recovery, mitigation of application stragglers, mitigating power fluctuation issues during large-scale ML/AI training by efficiently leveraging idle GPU resources for other computing tasks, and minimizing I/O interference in multi-CPU-GPU setups for multiple applications sharing an exascale high-performance computing infrastructure. Through its global view of I/O enabled by a push-based in-memory computing approach, AIMSS promises significant performance improvements for data-intensive applications by actively handling data movement, while eliminating the need for manual tuning and inefficient application-based data management.**

Index Terms—**in-memory active storage systems, streaming, ML/AI, HPC CPU-GPU, unified storage and compute, push-based streaming model**

I. Introduction: Motivation, Vision and Objectives

In today's data-driven world, exemplified by recent advancements in large language models (LLMs, e.g., OpenAI's GPT 4, Google Gemini) and new Cloud-HPC services (e.g., HPC federated learning [1]), the rapid growth of machine learning/artificial intelligence (ML/AI) and big data applications has generated an unprecedented demand for scalable, energy-efficient and fault-tolerant, data-intensive, and Active In-Memory Storage Systems (AIMSS) in support of ML/AI over large-scale HPC infrastructure including many-core CPU-GPU nodes, large memory clusters (TBs/node) and advanced interconnects (e.g., NVIDIA Infiniband).

Scaling AIMSS (as depicted in Figure 1) across the memory hierarchy of CPUs and GPUs [2] at HPC exascale (encompassing thousands of compute nodes with millions of CPU/GPU cores and hundreds of TBs of memory) presents a significant challenge. For example, GPUs can spend up to 70% of their time idle, waiting for data [3]. This inefficiency highlights the need for a more effective data management approach. Our primary focus is to enable the efficient execution of data-intensive workloads by introducing a novel programming model that allows applications to **delegate data movement** responsibilities to AIMSS. This delegation, achieved through a **push-based streaming execution programming model**, allows AIMSS to leverage application-specific data access (such as consumer offsets and read/write patterns) for optimizing data movement and execution across the HPC storage and computing infrastructure. As we argue in the next sections, this includes having application workloads completely delegate data movement tasks such as ingestion, output writing, and data shuffling.

Fault tolerance, a critical challenge at exascale, can lead to significant wasted compute capacity (20% or more) due to failures and recovery, as highlighted by the European strategic research agenda for HPC [4] [pages 79-82]. Addressing this challenge is a core focus of our research. Our second goal is to develop a fault-tolerant in-memory storage system for HPC exascale. AIMSS will achieve this through an immutable log-structured design and its novel push-based streaming programming model. Our envisioned approach enables valuable application insights into data access patterns, facilitating faster recovery, as detailed in the following sections.

Extreme power jitter [5] [section 3.3], arising from the synchronization of tasks like checkpointing [6], collective communication [7], and training computations during large-scale LLM training, presents a significant challenge (synchro-

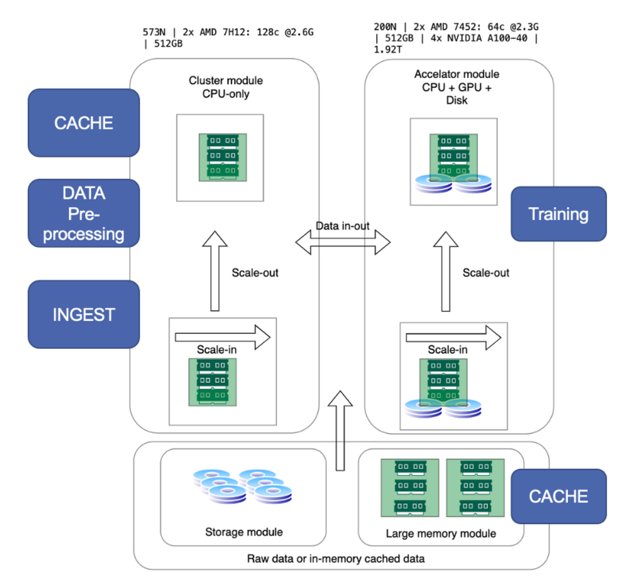

Fig. 1. The scalability challenge and hardware overview of main HPC MeluXina modules and how they fit various ML pipeline components.

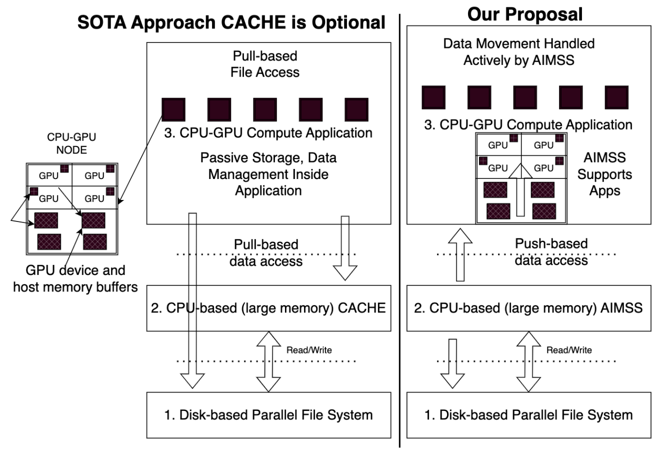

Fig. 2. The AIMSS active push-based storage approach versus passive pull-based state of the art (SOTA) storage approach. AIMSS manages resources depicted in blue in Figure 1 and integrates with LLM-engines for training through the push-based streaming execution model.

nized power fluctuations across tens of thousands of GPUs can strain data center power grids, potentially reaching tens of megawatts). To address this, our third goal is to mitigate, and potentially eliminate, this power jitter through a unified AIMSS and computing engine enabled by our proposed push-based streaming execution model. By leveraging AIMSS's awareness of remaining computations, derived from hints within data streams, we can efficiently utilize idle GPU resources during synchronization tasks such as checkpointing, which often takes tens of seconds. This active resource management, enabled by AIMSS's in-memory storage capabilities, promises to reduce power jitter by dynamically scheduling other tasks on otherwise idle GPUs.

Therefore, the central research question driving our envisioned AIMSS is: **How can we efficiently (in terms of energy, performance, and developer transparency to exascale deployments) scale in and scale out large-scale ML/AI pipelines on HPC infrastructure in a fault-tolerant manner?** This paper is structured as follows: Section II introduces the architecture and design principles of AIMSS. Section III presents a novel push-based stream-based programming model for AIMSS. Section IV explores the architectural optimizations enabled by AIMSS. Section V discusses related work and highlights AIMSS's vision for optimization contributions.

II. The AIMSS Approach

Our key insight is that closely integrating (in-memory) storage and processing for ML by delegating data movement control from the application to the storage layer, as our AIMSS proposes, will lead to various optimizations (explained later). Our global vision for AIMSS is a unified storage and compute architecture for ML/AI processing on HPC infrastructure, powered by a **push-based streaming execution model** with the following key benefits:

- **Unified CPU-GPU Deployment and Optimized Performance:** AIMSS will be deployed across CPU-GPU HPC infrastructure, leveraging their combined memory resources to support a push-based stream-based programming model (as depicted in Figure 2). This unified approach will facilitate efficient data movement and processing at HPC exascale.
- **Transparent Scalability and Resiliency:** AIMSS will provide users with transparency and resiliency while automatically scaling ML pipelines on HPC infrastructure. This will empower users to focus on their core research and development tasks without being burdened by the complexities of manual scaling and fault tolerance storage management.

AIMSS manages various data movement operations in support of processing engines that typically deploy pipelines of operators, including source and sink operators. Source operators fetch input data from a distributed storage system (e.g., disk-based file or caching systems) using a pull-based approach. Sink operators, on the other hand, write data to the storage system using a push-based approach. In addition, shuffle operators are responsible for redistributing data based on partitioning methods, such as key-based partitioning. Traditionally, the storage system plays a passive role, responding to read/write requests, while the shuffle mechanism is implemented at the application level. Managing source, sink and shuffle data movement, AIMSS will have a global I/O overview advantage over current approaches.

Recognizing that data-intensive applications (e.g., real-time streaming, LLM training) require continuous data movement (input, output, and shuffling), we propose the AIMSS strategy

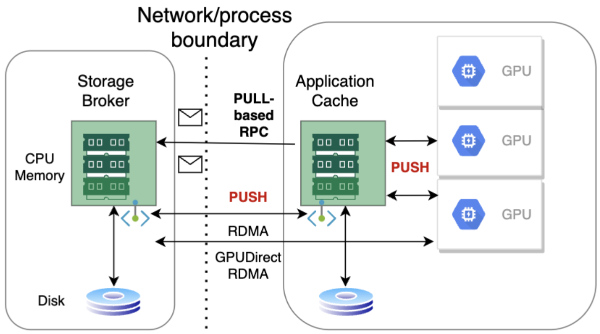

Fig. 3. The novel AIMSS architecture for scalable, unified storage and ML processing comprises coordinators (managing metadata and system components), CPU-based Storage Brokers, e.g., [8], interacting with file/cache systems to fill the CPU-GPU-based Application Cache components that are managing host and kernel memory for the application through push-based stream buffers, e.g., [9]. Unlike existing in-memory streaming storage systems, AIMSS uniquely targets both CPU and CPU-GPU memory, enabling optimized, unified data movement across heterogeneous compute resources.

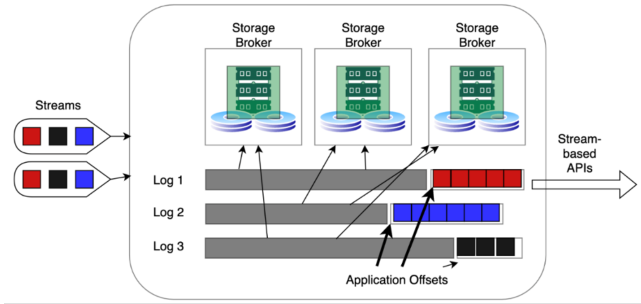

Fig. 4. Our novel approach to fast crash recovery for unified in-memory log-structured storage, e.g., [10], builds on AIMSS's push-based model, enabling recovery from the last recorded application consumer/producer offsets. This design allows for faster recovery compared to traditional methods that require full log recovery. By focusing only on relevant application stream offsets, AIMSS significantly reduces the overhead associated with restoring large-scale applications after crashes.

to delegate these operations to the AIMSS itself. Our strategy works as follows: Source operators register their input stream requirements (including any filtering functions) with AIMSS, which then proactively fills input buffers using a push-based approach. Sink operators operate on pre-registered stream buffers and notify AIMSS when data is ready to be written, triggering asynchronous persistence to disk and buffer reuse. Shuffle operations function similarly, allowing AIMSS to reorganize input stream buffers asynchronously.

Our vision of decoupling data movement operations from processing operators is realized through the AIMSS middleware layer that sits between the disk-based file storage system and application engines deployed on, for example, CPU-GPU nodes. Furthermore, AIMSS manages both CPU and GPU host memory and integrates with GPU device memory through native code (e.g., CUDA streams), using a push-based approach. Garbage memory collection is managed at the AIMSS level. All stream metadata is registered with AIMSS before and during deployment when Source, Sink, and Shuffle operations are delegated. When an application crashes or shuts down, AIMSS automatically cleans up its associated active streams.

Optimizing data movement for enhanced scalability, better performance, faster data and application recovery, and reduced power jitter at exascale is significantly more efficient when managed at the AIMSS data system level. While there are potential challenges for efficiently managing metadata for a massive number of streams, AIMSS contrasts with today's approach, where this burden often falls on application developers and their engines, leading to sub-optimal performance and increased complexity.

III. Our Proposal: A Push-Based Streaming Programming Model for Enabling AIMSS

The core concept of our vision is integrating stream phases for both read and write I/O-intensive operations, including shuffling. This integration provides valuable insights into application data access behavior, enabling the stream storage layer (i.e, AIMSS) to optimize I/O performance, minimize I/O interference, and enhance both checkpointing efficiency and recovery speed. By replacing traditional passive storage approaches with the AIMSS framework, application developers can unlock significant performance gains and benefit from transparent data management workflows (e.g., avoid tuning efforts). Storage system techniques can more efficiently develop better data movement optimizations compared to letting this effort on developers.

Execution model APIs. The following listing outlines the essential APIs for the AIMSS push-based streaming execution model. Compute engine's consumers and producers operators (e.g., GPU kernel tasks) create streams to interact with shared in-memory buffers managed by AIMSS through these APIs. Source, sink and shuffle operators delegate their read and write IO actions to AIMSS that is responsible to manage these push-based shared stream buffers.

```
CreateStream(ParentStreamId, InputSource,
    PartitionId, KernelReadWriteAttributes): return
    streamId # Initialize a new stream on AIMSS.
    Usage for read or write specified. Returns a
    unique streamId.
ReadFrom(StreamId) # Provides a set of shared
    StreamBuffers to iterate over
WriteTo(StreamId, StreamBuffer) # Write streamBuffer
    's content to a specified stream.
ShuffleStream(StreamId) # Signal shuffle ready.
ShuffleStreams(ParentStreamId) # Shuffle starts when
     all streamIds of ParentStreamId are ready
DestroyStream(StreamId)
```

AIMSS replaces the passive storage consumption model APIs. The traditional file-based storage model involves the application compute engine directly managing memory buffers and coordinating data movement. Unlike AIMSS, this approach requires the application to orchestrate tasks such as creating and reading files, writing data, shuffling files based on partition functions, and managing deletions. With AIMSS, the resulting overhead, including stream buffer management and data transfer coordination, can be minimized through strategies such as asynchronous data movement and efficient buffer reuse mechanisms.

IV. Data-based Architectural Optimizations Enabled By AIMSS

The rationale behind a push-based streaming model, in addition to its low-latency processing advantages (see [9]), stems from the continuous and bursty [5] data processing requirements of use cases like LLM training and real-time stream processing, which often involve petabytes of input data and checkpointing data with tens of TB/s peak throughput. By proactively managing data movement (input, output, and shuffling), a push-based stream storage system minimizes GPU idle time, reduces costs, and enables optimizations not easily achievable with a pull-based model, such as reduced I/O interference, faster recovery, straggler mitigation, and higher ingestion/checkpointing throughput. Moreover, the application knowledge insights provided by the push-based protocol's offsets eliminate the need for complex ML models and monitoring infrastructure to predict access patterns, simplifying the optimization process.

Traditionally, research engineers manually tune data partitioning, chunking, shuffling, checkpointing, and recovery during LLM training iterations. AIMSS, through the push-based model interactions, transparently manages these data movement operations on both consumer (input processing and recovery) and producer (shuffling, checkpointing) processes. Shifting data movement control from the application to the storage layer enables better optimization of ingestion and recovery due to its inherent application knowledge provided by stream access patterns available now to AIMSS.

Given the cost-effectiveness of CPU memory compared to expensive high-end GPUs, we advocate for aggressively optimizing data movement into and out of GPUs, leveraging AIMSS as a smart cache that provides needed storage features like availability and durability. Moreover, hardware trends (better interconnects [11], faster memory) will support our radical data movement approach. Beyond fast, scalable and dynamic data access, AIMSS provides two additional benefits: simplified fault tolerance implementation and the ability to detect stragglers easier (Application Cache nodes provide metadata of application compute tasks that may exhibit slower progress during training iterations).

As exemplified in Figure 3, AIMSS will address partial recovery by leveraging log-structured in-memory storage [12] and push-based data movement, potentially over RDMA technology [13].

A key technical implementation challenge lies in optimizing the pipeline of GPU kernel computations and push-based data feeding of GPUs. Leveraging dynamic stream partitioning and push-based data movement for processing, AIMSS provides a foundation for simplifying application scalability. GPU kernels create and manage streams (see the previous section APIs) by interacting with the colocated Application Cache.

Critical questions include determining suitable data dynamic partitioning and program parallelism mechanisms for efficiently feeding multiple GPUs, and how to cache these datasets to ensure applications are not delayed. Another aspect is designing and developing a push-based approach for CPU to CPU-GPU nodes integration and examining its trade-offs in terms of availability, partitioning, performance, and fault tolerance.

Current storage and processing systems often handle recovery independently, lacking the application-level insights needed for efficient recovery prioritization. AIMSS addresses this by leveraging stream offsets within its log-structured [10] storage design (illustrated in Figure 4). Upon a crash, AIMSS prioritizes recovering logs starting from the application's last consumed offsets, ensuring rapid access to the data needed for immediate application restart. This contrasts with traditional systems that perform full log recovery, incurring unnecessary overhead and delaying application restart.

V. Related Work

Enabling the unified AIMSS architecture requires the integration of several functional components. Data Ingestion [14] acquires, buffers, and temporarily stores in-memory fast data streams and raw file data. Data (persistent/caching) storage [15] provides durability, availability, and fault tolerance. Finally, big data processing [16] and ML/AI analytics [17], [18] enable ML/AI applications to efficiently consume data streams.

In contrast to monolithic architectures, e.g., [19], which can optimize data-related tasks more efficiently, these decoupled layered architectures do not easily benefit from such cross-layer optimizations. While each specialized component benefits from an open-source community, coupling these components in complex architectures often results in a trade-off between productivity and performance/cost efficiency. However, emerging approaches like push-based streaming integration, as demonstrated by KerA [20], offer a feasible path towards achieving both modularity and optimized data movement in unified architectures like AIMSS.

Apache Kafka [21], [22], a CPU-only cloud stream storage solution, will not scale at HPC exascale and it requires time-consuming and costly manual data re-partitioning, lacking support for partial recovery. AIMSS could build over our in-memory storage system KerA [8]–[10], [20], leveraging its dynamic partitioning and push-based streaming integration, although it currently lacks support for GPUs as argued in this paper. While fault-tolerant storage systems typically fully recover crashed nodes [23], they often do so without considering application-specific needs [24].

Notably, no existing system fully embodies the AIMSS active data movement approach. Consequently, efforts for straggler mitigation and multi-application I/O interference mitigation typically rely on resource-intensive monitoring tools, often introducing significant overhead. While AIMSS utilizes CPU memory to manage metadata and orchestrate data movement, the potential performance gains and reduced operational costs associated with its active approach can outweigh the expense of the additional CPU memory required.

As we previously argued in [9] for a push-based streaming model across the computing continuum, more recent research on data flow in modern hardware [25] also supports the concept of stream processing [26] across the entire architecture. However, while their focus is primarily on reducing data movement, and thus orthogonal to ours, AIMSS takes a distinct data movement approach to seamlessly integrate with and enhance existing processing engines. To ensure AIMSS's correctness and robustness (e.g., [27]), we plan to employ a holistic design approach enabled by TLA+ [28], [29], a language specifically designed for specifying and verifying concurrent and distributed systems.

VI. Conclusion

Our envisioned AIMSS, with its push-based streaming execution model, offers a compelling paradigm shift for managing data in exascale applications. By leveraging data immutability, application access patterns, and system-storage-level control over data movement, AIMSS promises to improve resource utilization, minimize GPU idle time, and simplify fault tolerance, while trading off CPU memory. By removing the burden of tuning data movement from application developers, AIMSS enables optimizations not easily achievable with traditional in-application-based methods. The design specification, implementation and evaluation of a unified AIMSS and ML/AI architecture will be the focus of our future work.

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] J. Hoffmann, P. Bauer, I. Sandu, N. Wedi, T. Geenen, and D. Thiemert, "Destination earth – a digital twin in support of climate services," *Climate Services*, vol. 30, p. 100394, 2023.

[2] S. Mittal and J. S. Vetter, "A survey of cpu-gpu heterogeneous computing techniques," vol. 47, no. 4, jul 2015. [Online]. Available: https://doi.org/10.1145/2788396

[3] S. Ben David, "Why a data plane architecture is critical for optimizing next-generation workloads," in *Proceedings of the 2022 Workshop on Emerging Open Storage Systems and Solutions for Data Intensive Computing*, ser. EMOSS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 9.

[4] T. E. T. P. for High Performance Computing (ETP4HPC), "Strategic research agenda for hpc in europe," 2022. [Online]. Available: https://www.etp4hpc.eu/pujades/files/ETP4HPC-SRA5_2022_web.pdf

[5] A. D. et al., "The llama 3 herd of models," 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[6] A. Maurya, R. Underwood, M. M. Rafique, F. Cappello, and B. Nicolae, "DataStates-LLM: Lazy Asynchronous Checkpointing for Large Language Models," in *HPDC'24: 33nd International Symposium on High-Performance Parallel and Distributed Computing*, Pisa (IT), Italy, Jun. 2024. [Online]. Available: https://hal.science/hal-04614247

[7] N. Dryden, N. Maruyama, T. Moon, T. Benson, A. Yoo, M. Snir, and B. Van Essen, "Aluminum: An asynchronous, gpu-aware communication library optimized for large-scale training of deep neural networks on hpc systems," in *2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC)*, 2018, pp. 1–13.

[8] 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 ICDCS*. IEEE, 2018, pp. 1480–1485.

[9] O.-C. Marcu and P. Bouvry, "In support of push-based streaming for the computing continuum," in *15th Asian Conference on Intelligent Information and Database Systems*, Phuket, Thailand, Jul. 2023.

[10] O.-C. Marcu, A. Costan, B. Nicolae, and G. Antoniu, "Virtual Log-Structured Storage for High-Performance Streaming," in *Cluster 2021 - IEEE International Conference on Cluster Computing*, Portland / Virtual, United States, Sep. 2021, pp. 1–11.

[11] M. Vuppalapati, S. Agarwal, H. Schuh, B. Kasikci, A. Krishnamurthy, and R. Agarwal, "Understanding the host network," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 581–594. [Online]. Available: https://doi.org/10.1145/3651890.3672271

[12] S. M. Rumble, A. Kejriwal, and J. Ousterhout, "Log-structured memory for dram-based storage," in *Proceedings of the 12th USENIX FAST*, ser. FAST'14. USA: USENIX Association, 2014, p. 1–16.

[13] S. Ma, T. Ma, K. Chen, and Y. Wu, "A survey of storage systems in the rdma era," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, p. 4395–4409, dec 2022.

[14] 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," INRIA Rennes - Bretagne Atlantique and University of Rennes 1, France, Technical Report RT-0501, Nov. 2018.

[15] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[16] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, p. 107–113, jan 2008.

[17] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 423–438. [Online]. Available: https://doi.org/10.1145/2517349.2522737

[18] M. e. a. Abadi, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USA: USENIX Association, 2016, p. 265–283.

[19] J. Zou, A. Iyengar, and C. Jermaine, "Pangea: Monolithic distributed storage for data analytics," *Proc. VLDB Endow.*, vol. 12, no. 6, p. 681–694, feb 2019. [Online]. Available: https://doi.org/10.14778/3311880.3311885

[20] O.-C. Marcu, "KerA: A Unified Ingestion and Storage System for Scalable Big Data Processing," Theses, INSA Rennes, Dec. 2018. [Online]. Available: https://theses.hal.science/tel-01972280

[21] G. e. a. Wang, "Consistency and completeness: Rethinking distributed stream processing in apache kafka," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2602–2613.

[22] A. e. a. Povzner, "Kora: A cloud-native event streaming platform for kafka," *Proc. VLDB Endow.*, vol. 16, no. 12, p. 3822–3834, aug 2023. [Online]. Available: https://doi.org/10.14778/3611540.3611567

[23] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in ramcloud," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 29–41. [Online]. Available: https://doi.org/10.1145/2043556.2043560

[24] A. Magalhaes, J. M. Monteiro, and A. Brayner, "Main memory database recovery: A survey," vol. 54, no. 2, mar 2021.

[25] A. Lerner and G. Alonso, "Data flow architectures for data processing on modern hardware," in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, May 2024, presented at the Data Engineering Future Technologies Track.

[26] O.-C. MARCU and P. BOUVRY, "Big data stream processing," University of Luxembourg, Tech. Rep., 30 August 2024.

[27] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Trans. Program. Lang. Syst.*, vol. 12, no. 3, p. 463–492, jul 1990. [Online]. Available: https://doi.org/10.1145/78969.78972

[28] L. Lamport, "The temporal logic of actions," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 3, p. 872–923, may 1994.

[29] B. Batson and L. Lamport, "High-level specifications: Lessons from industry," in *Formal Methods for Components and Objects*, F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 242–261.