XD/ML Pipelines: Challenges in Automated Experimental Science Data Processing

Justin M. Wozniak, Ryan Chard, Kyle Chard, Bogdan Nicolae, and Ian Foster
Argonne National Laboratory
woz, rchard, chard, bnicolae, foster @anl.gov

Topic: Data-management support for AI and complex workflows.

I. Challenge

Experimental data processing pipelines are growing not just in volume and velocity but also in complexity, as machine learning (ML) methods become an ingrained part of the workflow at multiple levels. Data access and transfer rates are often a limiting factor in overall time-to-completion. These workflows, here dubbed Experimental Data for Machine Learing (XD/ML) pipelines are also heterogeneous, consisting of phases that have different performance limits.

Typical XD/ML pipelines contain multiple phases relevant to the study of data management architectures. These include *data capture:* encompassing retrieval from the instrument, fast edge inference, and staging to temporary storage; *indexing:* extracting and synthesizing metadata, loading into catalogs and databases; *reconstruction:* moving selected instrument data to high performance computing (HPC) for full first-principles analysis; and *model training:* using reconstructed data to (re)train edge inference models. Each of these phases could involve a full read and write of the dataset to persistent storage, but many optimizations are typically employed. Additionally, in the presence of ML-based, application-aware, policy-based workflow decisions, these steps could proceed in different orders, be omitted, or re-executed in loops.

The data access workloads in complex XD/ML pipelines are qualitatively more difficult to diagnose, validate, optimize, and reason about when compared to traditional data acquisition patterns. This is not only because of the inherent difficulty of explaining ML models, but also because the ecosystem of data management tools and services is not tightly integrated. Ultimately, this prevents researchers from explaining why a particular result was obtained, sharing training data with others, or reproducing experiments [1].

II. Opportunity

Experimental science projects continue to advance from a manual "collect data, analyze later" paradigm to an automated, AI-enhanced, self-documenting "collect, analyze, feedback, document" paradigm. Project teams typically assemble one or more "flows" to perform a series of tasks, usually associated with particular available hardware. Over time, multiple flows are assembled into an application suite, generalized for multiple uses, and made portable to multiple data acquisition and processing systems. Scientific developers in these contexts, however, lack an ecosystem in which such workflows can be rapidly assembled, ported, scaled, and optimized; here we describe how the problem space can be bridged to a Management and Storage of Scientific Data (MSSD) context.

These flows have different computational requirements and priorities: for example, in serial crystallography experiments it is important that quality control and stills processing [2] be performed within seconds, and can employ local accelerators, while structure determination requires HPC but can be delayed if needed. Ptychography experiments that process intensive data streams on an HPC system, however, can introduce severe I/O bottlenecks due to, for example, the random sample loading used in model training to reduce bias. AI/ML-specific sample loading and caching optimizations are essential, with special emphasis on awareness of flow and data streaming. Additionally, flows may be controlled by nearly opaque ML models, making dynamic decisions about progress in the overall experiment, (e.g., when new data inferred to be of type D are produced, retrain model M, record outputs in catalog C, and proceed to the next sample material).

A. Exposing data access challenges through policy engines

Policy engines must be deployed at one or more levels to ease the interoperation of flows and regulate users and facilities. Such policy engines need not be centralized. A handful of key policy points are near-universally needed for flow management. These include: *deadline:* a flow must be started or completed by a particular deadline; *priority:* a flow should be prioritized relative to other flows from the same user, group, or facility; *resource limit:* a flow component should not consume more than a predetermined limit in terms of resource allocation used; *locality:* computation should minimize data transfer, favor use of an accelerator, or execute close to an instrument; *application-specific conditions:* for example, an ML model should be trained until it reaches a threshold metric. At the present time, teams are integrating ML methods into pipelines to minimize human intervention with the workflow, e.g., for automatic detection of regions of interest, mechanically cropping diffaction patterns, and enhancing reconstruction with AI/ML techniques.

Generalized flow policy management offers a potential interface between XD/ML researchers and the MSSD community. By exposing the structure of flows and exploiting MSSD optimizations, high-quality, portable, re-usable flows could be developed.

Expecting the emergence of a fully generalized, centralized policy engine is unreasonable. Rather, steps toward distributed policy management can be taken following the approach of stable distributed control systems [3]. This approach would start with the development of common performance control interfaces across services and tools. Then, local, constrained policy decisions and optimizations could be made. Meta-controllers could be deployed as applications are scaled up to negotiate broader problems, and optionally integrated with centralized policy services.

The goal of distributed service negotiation is not unreachable. Interoperable software and services are necessarily being developed in the HPC community. For example, the E4S [4] and Spack [5] projects have built up a very large suite of build routines enabling HPC teams to integrate and even link many tools together. Similarly, the ExaWorks project [6] is building an interoperable ecosystem of workflow tools that can build up nested workflows and other compositions.

B. Connecting flows to streams with data movement optimizations for continual learning

In this context, experimental science flows governed by policies that control data-in-motion introduce opportunities to optimize data management at multiple levels. In terms of abstractions, it is not sufficient to reason about input data as a static sequence of bytes available from the beginning (either as a file on parallel file system or a set of key-value pairs in an object store). Instead, data arrive continuously as a stream and computations need to be started as soon as possible, then adapted to process new data on-the-fly, as they arrive. This raises an entirely new set of challenges in the ML space. For example, a DNN model training cannot simply update the model by processing new mini-batches, as this leads to the problem of *catastrophic forgetting* (bias towards newest samples). On the other hand, accumulating all streamed data and retraining from scratch every time new mini-batches are available is not feasible either. Therefore, there is a need for advanced data streaming abstractions that look and act like normal streams, but can be augmented with policies to transparently mix historic and new data in order address application requirements.

In the example considered above, one can imagine data streams that automatically cache representative training samples based on a selection policy. Then, when a new mini-batch arrives, it is automatically augmented to include historic representative training samples, which effectively results in a transparent rehearsal for the DNN training without the need to modify the training process itself. Naturally, the design points of such abstractions have important implications on the whole storage stack, including: *locality:* where to cache and/or persist historic data; *categorization:* whether to handle live data and historic data separately or together; *hierarchy:* how to leverage multiple storage levels; *new hardware:* how to take advantage of persistent memory to reduce the performance gap between accessing historic data and live data.

C. Capturing, interpreting, and replaying automated data access decisions

Given that data access is supported by policy-level tools, it becomes possible to automate the inspection of flow progress and to make created datasets FAIR. Provenance records generated by workflows must be augmented with model version history so as to record how a possibly complex ensemble of variably-accurate models have been trained and updated over time, because these models impact the MSSD-relevant data accesses made by the system. The system would thus create a structure so the user can ask how a workload pattern was obtained, in the form of a provenance history.

III. Timeliness

The identified challenges and opportunities are highly relevant in the next 2-3 years. At Argonne, data flows between APS and ALCF are already being used by multiple applications. Without addressing these challenges, performance and scalability will be limited, because the benefits of advanced processing rates available on emerging exascale systems will be lost to inefficiencies in data movement. The Braid project at Argonne is addressing some of these challenges in deploying workflows based on Globus Flows [7] and the Braid Provenance Database [8].

References

[1] J. Borycz and B. Carroll, "Implementing FAIR data for people and machines: Impacts and implications - Results of a research data community workshop," 2020.

[2] M. Wilamowski, D. A. Sherrell, G. Minasov, Y. Kim, L. Shuvalova, A. Lavens, R. Chard, N. Maltseva, R. Jedrzejczak, M. Rosas-Lemus, N. Saint, I. T. Foster, K. Michalska, K. J. F. Satchell, and A. Joachimiak, "2'-o methylation of RNA cap in SARS-CoV-2 captured by serial crystallography," *Proceedings of the National Academy of Sciences*, vol. 118, no. 21, 2021. [Online]. Available: https://www.pnas.org/content/118/21/e2100170118

[3] E. Camponogara, D. Jia, B. Krogh, and S. Talukdar, "Distributed model predictive control," *IEEE Control Systems Magazine*, vol. 22, no. 1, pp. 44–52, 2002.

[4] M. Heroux, J. Willenbring, S. Shende, C. Coti, W. Spear, J. Peyralans, J. Skutnik, and E. Keever, "E4S: Extreme-scale Scientific Software Stack," in *Collegeville Workshop on Scientific Software*, 2020.

[5] T. Gamblin, M. P. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and W. S. Futral, "The Spack package manager: Bringing order to HPC software chaos," in *Proc. SC*, 2015.

[6] A. Al-Saadi, D. H. Ahn, Y. Babuji, K. Chard, J. Corbett, M. Hategan, S. Herbein, S. Jha, D. Laney, A. Merzky, T. Munson, M. Salim, M. Titov, M. Turilli, T. D. Uram, and J. M. Wozniak, "ExaWorks: Workflows for exascale," in *Proc. WORKS @ SC*, 2021.

[7] R. Chard, K. Chard, S. Tuecke, and I. Foster, "Software defined cyberinfrastructure for data management," in *Proc e-Science*, 2017.

[8] J. M. Wozniak, Z. Liu, R. Vescovi, R. Chard, B. Nicolae, and I. Foster, "Braid-DB: Toward AI-driven science with machine learning provenance," in *Proc. Smoky Mountains Conference*, 2021.