Braid-DB: Toward AI-driven Science with Machine Learning Provenance

Justin M. Wozniak, Zhengchun Liu, Rafael Vescovi,
Ryan Chard, Bogdan Nicolae, and Ian Foster

Argonne National Laboratory, Lemont IL 60439, USA
Corresponding author: woz@anl.gov
WWW home page: https://github.com/ANL-Braid

Abstract. Next-generation scientific instruments will collect data at unprecedented rates: multiple GB/s and exceeding TB/day. Such runs will benefit from automation and steering via machine learning methods, but these methods require new data management and policy techniques. We present here the **Braid Provenance Engine (Braid-DB)**, a system that embraces AI-for-science automation in how and when to analyze and retain data, and when to alter experimental configurations. Traditional provenance systems automate record-keeping so that humans and/or machines can recover how a particular result was obtained—and, when failures occur, diagnose causes and enable rapid restart. Related workflow automation efforts need additional recording about model training inputs, including experiments, simulations, and the structures of other learning and analysis activities. Braid-DB combines provenance and version control concepts to provide a robust and usable solution.

Keywords: provenance, machine learning, version control, database

1 Introduction

Modern science must increasingly deal with high data rates and volumes; as a consequence, machines are supplementing humans as decision makers: for example, concerning how and when to analyze and retain data, and when to alter experimental configurations. Thus it becomes essential to automate also record keeping so that humans and/or machines can determine how a particular result was obtained—and, when failures occur, diagnose causes. Such records must frequently be recursively structured, for example when raw data are supplemented by ML model predictions that depend on training data. These factors, plus highly automated, dynamic, and adaptive execution, make capture of provenance information essential. We need methods for capturing sufficient information about all data products produced, both to *enable inspection of workflow progress* and to *make data findable, accessible, interoperable, and reusable* (FAIR).

Interpreting such automatic decisions to understand progress or performance and to validate scientific results will require new provenance concepts and systems to capture not just the data obtained, but also the models produced, which may be more important in the era of artificial intelligence (AI) for science.

Traditional provenance systems automate record-keeping so that humans and/or machines can recover how a particular result was obtained—and, when failures occur, diagnose causes and enable rapid restart. Such systems are typically built to handle static workflow patterns and capture predictable forward progress.

The Braid Provenance Engine (Braid-DB), introduced herein, provides specific features for use in an environment dominated by externally-produced data, such as experiment instruments, and machine learning (ML) data access patterns, as depicted in Figure 1. It develops recursive and versioned provenance structures to capture how models may be constructed via other models (e.g., estimators and surrogates) and frequent model updates, allowing the user to track past decisions as models make decisions and are retrained. This paper will also survey partner applications for more detailed use cases and requirements. Braid-DB is a component of a larger effort within Argonne's Braid project to support experimental science workflows.

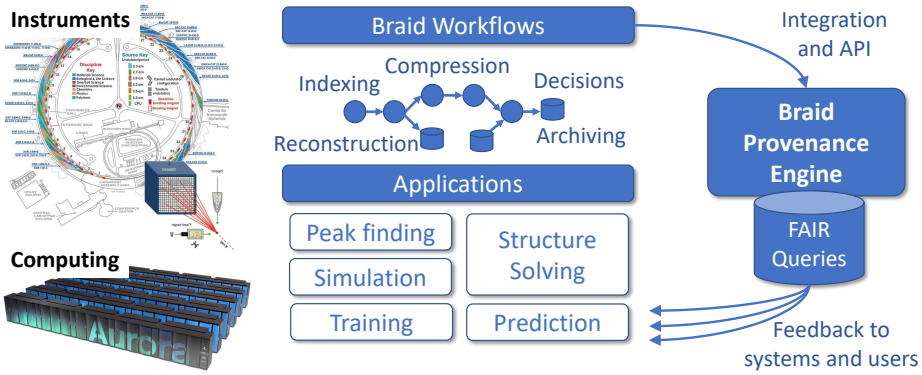

Fig. 1. Braid workflows and their connection to Braid-DB. Center top shows an example Braid workflow with various actions. The Braid provenance engine captures events from Braid tools or its API and provides a FAIR interface for the provenance of data products in the context of experiment, simulation, and learning.

Organization. The remainder of this paper is as follows. In §2, we provide background on provenance systems and other related efforts. In §3, we describe our approach in more detail, with its intended benefits. In §4, we describe the software architecture of our system. In §5, we outline several motivating case studies in experimental science that drove our design. In §6, we measure the performance of our system on a workload meant to stress all of its capabilities. In §7, we outline future work. In §8, we summarize the paper and provide concluding comments.

2 Background

Computer scientists have developed many provenance systems and related tools [20]. There is a significant body of prior work in provenance for scalable computing workflows, but additional developments are needed to address the needs of AI for Science workflows coupled with experimental science infrastructure. The need to capture not only data and associated metadata but also derived models in ways that are Findable, Accessible, Interoperable, and Reusable (FAIR) [26] is also highly relevant to this topic.

2.1 Provenance needs in AI for Science

A 2020 DOE AI for Science report [22] offered a broad depiction of the future of AI-supported scientific studies. Data management and provenance were identified as critical to enabling real-time adaptation in simulation and experiment workloads. The report noted the "need for enhancements in data storage, access, and management that would facilitate rapid identification of relevant data, transformations between different data representations, and capture of relevant provenance to assist in reproducibility of results." The report also noted that provenance is critical to making AI/ML workflows effective when coupling experiment, AI/ML, and HPC, including for the purpose of managing uncertainty and validation in results.

Another DOE report, on Data and Models for AI [10], identified provenance as a critical component of the AI for Science effort. The report noted that "[w]hen used for control or autonomous decision making in a scientific workflow, the trained model may be an important digital artifact for reproducibility of the results." This need to capture trained models is precisely why the conception of provenance needs to be expanded in the the context of ML and experiment-in-the-loop computing efforts. More broadly, the report considered the challenges faced by scientists attempting to use popular tools for scientific problems, and the need to manage data in AI for Science contexts in ways that satisfy FAIR principles. The report also stressed the importance of human oversight of science workflows; as we describe in the following, the Braid-DB effort stresses human interpretability as a key driver.

A report on Scientific Machine Learning [5] stressed the importance of changes to the types of scientific questions that can be asked of computing systems. The report considered the question of how to perform data acquisition, a mode of operation which creates complex data provenance structures. The report also noted the importance of being able to interpret results of ML-driven studies. Another report [8] noted the difficulties of creating data that are useful for humans and machines, and of defining and measuring FAIR.

Overall, the AI for Science paradigm emphasizes the need for increased automation at all levels of experimental science and the resulting need to reuse data analysis techniques and systems in new ways.

2.2 Developments in provenance concepts for machine learning

Traditional provenance models are designed to capture key aspects of a dataset, including its derivation history from other data, quality, and ability to be replicated or reproduced [20]. The derivation history of a dataset is a description of the computational processes that produced it, and the input data consumed by those processes. It is generally assumed that provenance captures a coarse-grained view of dataset production; fine-grained information such as delineating particular predictions in a model are not captured. Conventionally speaking, we are interested in capturing provenance data at a file and version level, not at the record level, to reduce the overheads associated with storing records about such small data.

The Open Provenance Model [17] defines provenance concepts without regard to the underlying technologies or systems used to represent them. These concepts include "artifacts", or data objects in the system; "processes", actions caused by artifacts that produce new artifacts; and "agents", the contexts that control processes. For example, artifacts could be files, processes could be running programs, and agents could be computer systems running programs and managing files. These concepts are connected with various kinds of edges that represent various types of dependencies. For a particular workflow, the result is an *OPM graph* that captures the causal dependencies among all data products.

Souza et al. [21] considered provenance management for ML workloads in sciences. They defined human actors in the ML work cycle, and integrated model training data and other ML-specific concepts, such as ML hyperparameters, into the provenance model. This study, however, did not consider the evolving versions of an ML model that may be produced during an iterative study, in which different ML predictions are produced over time by varying versions of a model. This is a key contribution of the Braid-DB model and is critical for the integration of provenance techniques in experiment-oriented computing.

Polyzotis et al. [18] similarly identified human factors as critical in the ML pipeline. This study considered aspects of raw data manipulation needed for ML training, including cleaning and enriching datasets for training. The paper also touches on privacy-sensitive and limited-access datasets. Metadata tracking schemes for ML workloads have been proposed previously. In ModelDB [24], a ML metadata and training data architecture is easily accessible from within ML-oriented programming environments, including a graphical front-end. Notably, ModelDB, like our Braid-DB, uses a branching history model to track changes over time. An Amazon prototype [19] develops a formal database schema for tracking models and their training data, based on previous but more narrowly defined models [15]. This schema allows for the creation of graphs of data transforms in the ML workflow. Neither system, however, explicitly tracks versions of models in model-model interactions or after iterations of model-experiment iterations. Additionally, neither explicitly mentions the hardware used for ML training and inference, an increasingly important factor for flows that include deep learning and heterogeneous systems [12,2].

2.3 Globus Flows

The Braid project within which Braid-DB is developed uses *Globus Flows* to describe and execute workflows [3]. This cloud-based service is designed to automate various data management tasks such as data transfer, analysis, and indexing. Flows are defined with a JSON-based state machine language which links together calls to external *actions*. Flows implements an extensible model via which external *action providers* (e.g., for transferring data with Globus Transfer, executing functions with funcX, associate identifiers via DataCite) can be integrated by implementing the Globus Flows action provider API. Flows relies on Globus Auth [23] to provide secure, authorized, and delegatable access to user-defined flows and to action providers.

3 Approach

We now describe how the Braid Provenance Engine capture various aspects of the computing-experiment loop for inspection and analysis.

3.1 Contributions

Our work in this area is motivated by two goals: to enable inspection of workflow progress and to produce workflow data products that are FAIR. To this end, we design Braid-DB data structures to be comprehensive, by which we meant that they capture information about not only those outputs that are eventually associated with publications, but also intermediate products that may or may not ultimately prove to be important [9,16]. Braid-DB captures metadata about not only static data, such as well-established experimental data, but also live experimental data, which are treated separately before being promoted to static data.

Braid-DB, like traditional provenance systems (§2), captures simulation inputs and outputs and model training inputs and outputs. Importantly, it integrates provenance records with model version history so as to record how a possibly complex ensemble of variably-accurate models were trained and updated over time. As training data may come from experiment, simulation, or other models (e.g., a higher-accuracy, higher-cost model, or an ensemble of lower-cost models) Braid-DB includes structures that allow a user to ask how a model inference result was obtained, in the form of a defensible *statement*. The provenance structure is designed to be portable so that provenance histories can be merged with other records from collaborating teams, as when a user borrows a model from another team.

We intended that the continuous generation and collection of records concerning inputs, code versions, locations, progress, and outputs of both individual actions and complete flows enable the construction of externally visible dashboards for monitoring application progress, and tools for tracking and reporting on, for example, the locations and sizes of data produced to date. These same

records can also be leveraged to synthesize metadata records for loading into catalogs. The system is designed to integrate with external capabilities such as those provided by the Materials Data Facility [6] and associated data ingest tools [7].

Realizing our goal of policy-driven automation demands machine-level understanding of flows (§2.3). We investigate what metadata representations and relationships are needed to maintain this understanding so as to enable automatic decisions, such as those to be made by workflow policy engines that are being developed within the Braid project (§4). Second, analyses often involve ML modules that are updated continuously by new data from multiple sources. This is a challenging provenance and version control problem. Thus in designing Braid-DB, we consider how to mix and match concepts from the provenance literature (§2) with popular version control concepts to provide a robust and usable solution to this model management problem.

3.2 Provenance structure

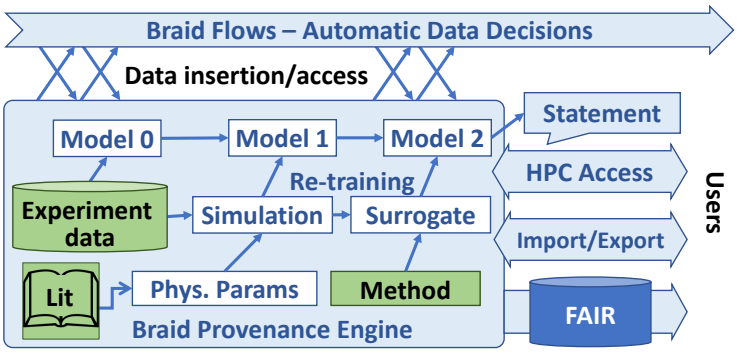

Fig. 2. Braid provenance internals and interfaces. Static records include experimental observations, data from literature, and algorithm definitions. Updateable records include simulation runs, surrogates, and models derived from other records. Small arrows indicate data accesses recorded as provenance information (e.g., model inputs).

We design Braid-DB data structures and implementation to enable automatic generation of identifiers [9,11], collection of descriptive metadata, and construction of provenance records to enable reproducibility, post-mortem analysis, and auditing of computations. We show the provenance structure of our system in Figure 2. Inside the "Braid Provenance Engine" box, all boxes are Braid Records, and the arrows indicate Braid Dependencies. At the base of the structure, shown in green, are static Records, such as quantities from the literature, outputs from physical experiment runs, and methods encoded in software. The objects included in the Braid-DB model include:

1. **BraidRecord:** A super-class for Braid-DB provenance records. Each such entity has a unique ID, a (possibly not unique) string name, a list of dependencies, and a dictionary of user-specified, string-keyed metadata tags,
2. **BraidFact:** A simpler object consisting of static data: for example, pre-existing trusted data or software, etc. BraidFacts may have a provenance outside the Braid-DB system.
3. **BraidData:** The Braid-DB representation of traditional provenance-tracked data, with traditional conceptions of its derivation history from other BraidData and/or BraidFacts. A Braid-DB containing only BraidData and BraidFacts would be functionally indistinguishable from a traditional provenance database.
4. **BraidModel:** An ML model tracked by Braid-DB. A BraidModel has the additional capability **update()**, which represents model exposure to other BraidRecords, possibly including other models. This includes the possibility of dependency cycles that capture complex interactions among models and data as experiment workflows progress.

4 Architecture

We now describe the software implementation of the Braid-DB system. The software-level goals of Braid-DB are to develop a system that implements the provenance structure described in §3 in a way that is easy to use and integrate with existing systems, and thus to provide a toolkit that can be rapidly adopted, shared, and extended.

4.1 Software performance targets

We design the Braid-DB prototype to be flexible and compatible with existing techniques and file formats, rather than a prescriptive framework. An HPC access module will be available via a MPI4Py library so that records can be rapidly ingested during an HPC run, and later merged with other records; this module will be used by our HPC features. The system will be scalable to the needs of an ML training-based workload. For reference of scale, we consider a challenge problem workflow from the ECP CANDLE project [28]. In this workflow, one training epoch is completed per node every five minutes. On an exascale system of 10K nodes, that is 33 records/second. Over one week of facilities experiment time, that would record 20M records. Achieving these rates will be the target of our database and HPC module.

We record metadata from Braid Flows in the provenance component in the form of structured metadata regarding data flows as they relate to objects represented in external storage, that is, reads and writes to/from flows. Thus, the provenance of an flow object in the store is the metadata history of the flows that have affected it, and so on.

4.2 Software components

We show the principal components of the Braid-DB implementation in Figure 3. At the top level, **applications and/or workflow systems** drive usage of Braid-DB. These components could include application workflows written using workflow systems such as Parsl [4] or Swift/T [27], which make it easy to call directly into Python libraries. Scientific applications and analysis systems such as NeXpy [1] could also access the system directly. A shell script library will also be provided to support such scripts.

Supporting the top layer are the **HPC cache** and **import/merge tools**. The *HPC cache*, a system component to support scalable Braid-DB workloads on exascale-class machines, will use a combination of database operation forwarding and aggregation to prevent overloading the underlying databases during bursty workloads. The *import/merge tools* will enable data slicing and extraction from Braid-DB instances so that subsets of data can be shared with others. These tools will also allow users to import external database records as a basis for the provenance of future experiments, to support use cases that involve reproducing and/or extending other experiments.

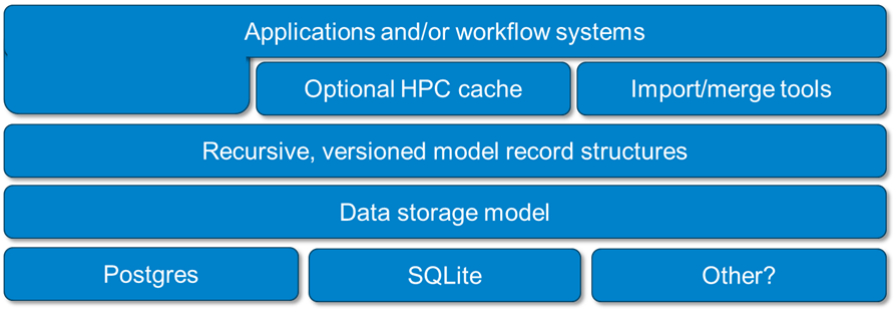

Fig. 3. Software architecture of the Braid Provenance Engine.

The **recursive, versioned model record structures** implement the abstraction described in §3. They provide Python access to Records and Dependencies and allow for these objects to be easily created, searched, loaded, manipulated, and stored. The **data storage model** persists the record structures in a non-volatile, consistent way, mapping Records and Dependencies, along with typical metadata such as timestamps and user tags, into a database schema. Our current implementation uses a standard SQL-based approach for capturing this information. In this model, given a record ID, its dependencies and other metadata can quickly be traced back to original static Records with minimal complexity.

4.3 Software implementation

We have developed a Braid-DB prototype [29] as an object-oriented Python library in which Braid-DB objects such as Records and Dependencies can be constructed, manipulated, and persisted to a data storage backend. A high-level API allows existing workflows to construct these objects persistently with just one or two additional lines of Python code.

We are developing methods for capturing sufficient information about all data products produced by flows to enable regeneration in supported workflow systems. To this end, we are working to combine existing and new capabilities to architect auto-documentation methods that collect provenance data automatically, with automated recording of identifiers, descriptive application-level metadata, and data access and dependency records to enable both reproducibility and post-mortem analysis and auditing of computations. Open questions include how to associate sufficient identifiers and metadata with the data, code, and resources involved in a computation, including dynamically updated ML models, automatically from workflow systems. The broader Braid project will orchestrate data flows using modular actions and integrate them with HPC resources, while enforcing data policies, e.g., to data capture quality and performance.

5 Case studies

We briefly describe two of the experimental science case studies that are guiding Braid-DB development, and one synthetic Braid-DB application workflow that we are using to evaluate the performance of Braid-DB implementations.

5.1 Provenance flow capture for training DNNs in x-ray science

Extremely high data rates at modern synchrotron and X-ray free-electron laser light source beamlines motivate the use of ML methods for data reduction, feature detection, and other purposes. Regardless of the application, the basic concept is the same: data collected in early stages of an experiment, data from past similar experiments, and/or data simulated for upcoming experiments are used to train a deep neural network (DNN) model that, in effect, learns specific characteristics of those data; this model is then used to process subsequent data more efficiently than would general-purpose models that lack knowledge of the specific dataset or data class [13]. In many cases, the DNN needs to be updated (retrained and fine-tuned) frequently to keep up with changes in experiment setup and sample conditions. Thus, a key challenge is to train models with sufficient rapidity that they can be deployed and applied within useful timescales. There are two common approaches to rapid DNN training: 1) using more powerful systems, such as purpose-built AI accelerators: e.g., TPU, Cerebras, SambaNova, GraphCore; and 2) use new data to fine-tune a similar model trained in the past.

Although purpose-built AI accelerators can train ML models much more rapidly than the computing clusters that may be deployed within an experiment

facility, such accelerators must commonly be deployed within a data center due to their cooling, power supply, ventilation, and fire suppression requirements. Thus, a distributed workflow is needed to automate DNN training with remote AI systems deployed at data center. The workflow commonly comprises the following six basic operations: 1) Collect a datum; 2) Simulate an experiment to generate a datum, d, without an experiment; 3) Analyze the datum using a conventional algorithm (e.g., Bragg peak extraction), generating an analysis (e.g., Bragg peak locations) [14], a; 4) Train (or retrain) a ML model with some number of $\{d, a\}$ pairs, generating a new model, m; 5) Deploy the new model m on an edge-AI device; and 6) Apply the model m to a new experimental datum, generating an estimated analysis, $\hat{a}$.

Provenance information is needed for such geographically distributed workflows in order to make their performance interpretable ("on what basis did we conclude that experimental data record r included feature $f1$?") and to enable troubleshooting ("why did we not detect feature $f2$ in r?"). Another straightforward use for provenance records is to support locating a pre-trained model in the repository to be fine tuned with new data. To support such applications, we need to record DNN models as they are generated, along with their training dataset, the convergence curve of the training process, and validation performance. Each model trained by the workflow must be discoverable by using information about its training, such as the training data and experiment metadata. Similarly, users must be able to query a model in the workflow history and obtain its training data and parameters.

5.2 Serial synchrotron x-ray crystallography

Serial synchrotron x-ray crystallography (SSX) enables novel studies of protein and enzyme dynamic processes by imaging small crystal samples 1-2 orders of magnitude faster than traditional crystallography techniques. SSX offers new opportunities for biologists to manage sensitive conditions such as time resolution by using light activation for change in conformation or change in pH, very low x-ray doses for sensitive samples, maintaining room temperature for more biologically-relevant environments, controlling radiation sensitivity for metalloproteins, and observing redox potentials in active sites. However, the increased data collection rates in SSX make it untenable for humans to manage experiments, because individual samples often result in hundreds of thousands of distinct images that must be analyzed, organized, and cataloged to deduce protein structures. Further, solving a protein structure is dependent on the configurations and thresholds used during the analysis and refinement process, necessitating fine-grained provenance tracking to associate structures with the raw data and analysis inputs used to create them.

At Argonne's Structural Biology Center at the Advanced Photon Source (APS) scientists have developed a Braid-compatible pipeline to process raw data, catalog and report interim results, and attempt to refine and solve protein structures [25]. This process captures sample information (including protein, preparation technique, exposure, and temperature) and feeds it into the analysis and

publication pipeline. We are now working to extend these efforts to capture and record fine-grained provenance information regarding the sample, beamline configuration, data, analysis inputs, and results, in order to track how specific protein structures are derived during an experiment. The development of a complete Braid workflow for the experiment will then simplify the data acquisition and processing of new samples by tuning the analysis parameters based on previous experiments. Furthermore, it will allow the experiment control algorithms to decide what are the next steps to complete the acquisition, for example, acquire more data and/or move on to the next sample.

5.3 The Mascot workflow

Application workflows such as the two just described tend both to use just a subset of Braid-DB capabilities and to incorporate numerous application complexities. Thus to illustrate Braid-DB capabilities and to permit flexible testing and performance evaluation (§6), we have developed the **Mascot workflow**, which exercises all Braid-DB capabilities in a synthetic setting.

The Mascot workflow starts with C configurations, each representing a static data record such as an instrument control script or some initial model training data. Each such configuration is stored in Braid-DB. Then, M models are instantiated, each depending on one configuration. These may be any kind of ML model that consumes training data and makes inferences. Then, N experimental cycles are run, each consisting of E experiment runs. Each run simulates the use of an experimental instrument that consumes configuration data and produces experiment data records. After each cycle, each model is exposed to all of the new experiment data, producing an updated model version and new provenance records, all of which are stored in Braid-DB. Each data record in the system, a configuration $c \in [0, C-1]$ or experiment $e \in [0, E-1]$ produces U URIs, representing an external data item referred to by the system.

A specific Mascot workflow can thus be configured by specifying the nature of the ML model used and by varying the values of the paraneters C, M, N, E, and U.

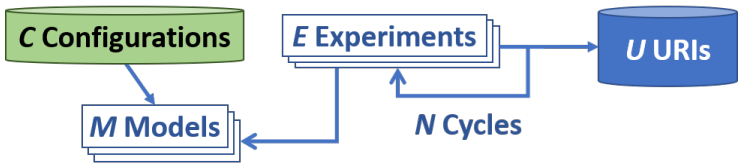

Fig. 4. Simplified depiction of Mascot workflow.

6 Performance

We describe an initial performance study of the Braid-DB system implementation (§4). In this study, we ran the Mascot workflow on a local workstation with an Intel i7-8700 running Ubuntu 20.04.0 and Python 3.8.2, using the Braid-DB SQLite backend. The Mascot workflow was configured to run using the parameterization described in §5.3. The workflow swept over a range of model counts $M \in [10, 100]$ and experiment counts $C \in [10, 100]$ with step 10. For each run, we fixed $M = C$, and set $N = 5, U = 3$.

Each experiment ran on a fresh SQLite database file. We ran two batteries of tests, one with SQLite automatic commits enabled and one with that feature disabled. We measured execution times internally by Python and counted the number of database entries (SQL rows) generated to obtain a database insert rate. Each (count, time) pair was run for five trials and the average was computed.

Fig. 5. Performance results for data insertion workflow

The results, plotted in Figure 5, show that the database insert rate for runs without commits converges to just under 100 000 inserts per second (the maximum is 97 166). The run with commits enabled reached a maximum of only 128 inserts/second. Clearly, it will be infeasible to rely on SQLite to handle concurrency in this mode, and we will use the integration with the workflow systems to ensure consistent multiprocess access to the database, as has been done with prior Swift/T applications.

Additionally, we created a Braid-DB instance with 20 146 000 entries by setting $E = 4000$ and $M = 1000$. Constructing this database with the Mascot workflow took only 169.9 seconds, and thus indicates that the performance goal based on the CANDLE example (§4.1) is easily achievable at this level of functionality. As we move forward we intend to maintain conformance with these rate targets as we add additional features and interfaces.

7 Future work

We have reported here on just the initial aspects of the development of a comprehensive system for managing provenance data from self-driving experiments. Remaining challenges include:

1. Deeper integration with experiment management systems
   (a) Capturing experiment-level details about conditions
   (b) Ability to replay experiment conditions and alternative scenarios
2. Deeper integration with data systems
   (a) Capturing resource availability and pressures
   (b) Ability to evaluate scenarios under different loads and system-level performance capabilities
3. Deeper integration with learning modules.
   (a) Capturing why decisions were made
   (b) Capturing available alternatives to decisions that were made
4. Deeper integration with scientific software abstractions
   (a) Investigating potential portable provenance abstractions for a range of scientific software
   (b) Developing APIs for use by a wider range of programming models, including zero-programming-effort integration via interception of other activity.

8 Conclusion

We have argued that modern science workflows require new provenance methods to meet the needs of machine learning-driven experimental science. Autonomous experimental science experiments challenge traditional approaches to experiment workflow reproduction and interpretation. New provenance structures are needed to capture not just the progress of workloads mandated by the software developer or experimental user, but also the choices made by ML modules, which may be based on subtle data variations or changes in underlying computing resources.

The development of ever-increasing automation poses new questions about the role of humans in experimental science. Initially, the goal of experimental automation will be to enable experimentalists to focus on higher-level problems in these workflows. In these cases, provenance will allow users to correct and improve experimental studies more quickly. Forward-looking replay and scenario evaluation will maximize the value obtained from time consumed on valuable infrastructure. A future goal will be to enable human users to specify much higher-level scientific questions and receive justifiable answers. The goals of the Braid Provenance Database is to accelerate progress toward these high-level question-and-answer specifications.

References

1. NeXpy: A Python GUI to analyze NeXus data. http://nexpy.github.io/nexpy
2. Abeykoon, V., Liu, Z., Kettimuthu, R., Fox, G., Foster, I.: Scientific image restoration anywhere. In: IEEE/ACM 1st Annual Workshop on Large-scale Experiment-in-the-Loop Computing (XLOOP). pp. 8–13. IEEE (2019)
3. Ananthakrishnan, R., Blaiszik, B., Chard, K., Chard, R., McCollam, B., Pruyne, J., Rosen, S., Tuecke, S., Foster, I.: Globus platform services for data publication. In: Proceedings of the Practice and Experience on Advanced Research Computing, pp. 1–7 (2018)
4. Babuji, Y., Woodard, A., Li, Z., Clifford, B., Kumar, R., Lacinski, L., Chard, R., Wozniak, J.M., Foster, I., Wilde, M., Katz, D.S., Chard, K.: Parsl: Pervasive parallel programming in Python. In: Proc. HPDC (2019)
5. Baker, N.: Basic research needs workshop for scientific machine learning, core technologies for artificial intelligence (2019)
6. Blaiszik, B., Chard, K., Pruyne, J., Ananthakrishnan, R., Tuecke, S., Foster, I.: The Materials Data Facility: Data services to advance materials science research. Journal of Materials **68**(8), 2045–2052 (2016)
7. Blaiszik, B., Ward, L., Schwarting, M., Gaff, J., Chard, R., Pike, D., Chard, K., Foster, I.: A data ecosystem to support machine learning in materials science. MRS Communications **9**(4), 1125–1133 (2019)
8. Borycz, J., Carroll, B.: Implementing FAIR data for people and machines: Impacts and implications - Results of a research data community workshop. Information Services & Use **40**(1-2), 71–85 (2020)
9. Chard, K., D'Arcy, M., Heavner, B., Foster, I., Kesselman, C., Madduri, R., Rodriguez, A., Soiland-Reyes, S., Goble, C., Clark, K., et al.: I'll take that to go: Big data bags and minimal identifiers for exchange of large, complex datasets. In: International Conference on Big Data (Big Data). pp. 319–328. IEEE (2016)
10. Fagnan, K., Nashed, Y., Perdue, G., Ratner, D., Shankar, A., Yoo, S.: Data and models: A framework for advancing AI in science. Report of the Office of Science Roundtable on Data for AI (2019), https://www.osti.gov/servlets/purl/1579323
11. Juty, N., Wimalaratne, S.M., Soiland-Reyes, S., Kunze, J., Goble, C.A., Clark, T.: Unique, persistent, resolvable: Identifiers as the foundation of FAIR. Data Intelligence pp. 30–39 (2020)
12. Li, J., Zhang, C., Cao, Q., Qi, C., Huang, J., Xie, C.: An experimental study on deep learning based on different hardware configurations. In: 2017 International Conference on Networking, Architecture, and Storage (NAS). pp. 1–6. IEEE (2017)
13. Liu, Z., Ali, A., Kenesei, P., Miceli, A., Sharma, H., Schwarz, N., Trujillo, D., Yoo, H., Coffee, R., Herbst, R., et al.: Bridge data center AI systems with edge computing for actionable information retrieval. arXiv preprint arXiv:2105.13967 (2021)
14. Liu, Z., Sharma, H., Park, J.S., Kenesei, P., Almer, J., Kettimuthu, R., Foster, I.: BraggNN: Fast x-ray Bragg peak analysis using deep learning. arXiv preprint arXiv:2008.08198 (2020)
15. Machine Learning Schema Community Group: W3C machine learning schema (2017), https://github.com/ML-Schema/
16. Madduri, R., Chard, K., D'Arcy, M., Jung, S.C., Rodriguez, A., Sulakhe, D., Deutsch, E., Funk, C., Heavner, B., Richards, M., Shannon, P., Glusman, G., Price, N., Kesselman, C., Foster, I.: Reproducible big data science: A case study in continuous fairness. PloS one **14**(4), e0213013 (2019)

17. Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephank, E., Busschef, J.V.: The Open Provenance Model core specification (v1.1). Future Generation Computer Systems **27**(6), 743–756 (2011)
18. Polyzotis, N., Roy, S., Whang, S.E., Zinkevich, M.: Data management challenges in production machine learning. In: 2017 ACM International Conference on Management of Data. p. 1723–1726. SIGMOD '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3035918.3054782, https://doi.org/10.1145/3035918.3054782
19. Schelter, S., Böse, J.H., Kirschnick, J., Klein, T., Seufert, S.: Automatically tracking metadata and provenance of machine learning experiments. In: Machine Learning Systems Workshop at NIPS (2017)
20. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. ACM Sigmod Record **34**(3), 31–36 (2005)
21. Souza, R., Valduriez, P., Mattoso, M., Cerqueira, R., Netto, M., Azevedo, L., Lourenço, V., F. de S. Soares, E., Melo, R., Brandão, R., Salles Civitarese, D., Vital Brazil, E., Ferreira Moreno, M.: Provenance data in the machine learning lifecycle in computational science and engineering. In: Workshop on Workflows in Support of Large-Scale Science at SC. pp. 1–10 (11 2019). https://doi.org/10.1109/WORKS49585.2019.00006
22. Stevens, R., Nichols, J., Yelick, K.: AI for Science Report on the Department of Energy (DOE) Town Halls on Artificial Intelligence (AI) for Science (2020)
23. Tuecke, S., Ananthakrishnan, R., Chard, K., Lidman, M., McCollam, B., Rosen, S., Foster, I.: Globus Auth: A research identity and access management platform. In: 12th International Conference on e-Science. pp. 203–212. IEEE (2016)
24. Vartak, M., Subramanyam, H., Lee, W.E., Viswanathan, S., Husnoo, S., Madden, S., Zaharia, M.: ModelDB: A system for machine learning model management. In: Proc. Workshop on Human-In-the-Loop Data Analytics. HILDA '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939502.2939516, https://doi.org/10.1145/2939502.2939516
25. Wilamowski, M., Sherrell, D.A., Minasov, G., Kim, Y., Shuvalova, L., Lavens, A., Chard, R., Maltseva, N., Jedrzejczak, R., Rosas-Lemus, M., Saint, N., Foster, I.T., Michalska, K., Satchell, K.J.F., Joachimiak, A.: 2'-o methylation of RNA cap in SARS-CoV-2 captured by serial crystallography. Proceedings of the National Academy of Sciences **118**(21) (2021). https://doi.org/10.1073/pnas.2100170118, https://www.pnas.org/content/118/21/e2100170118
26. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.W., da Silva Santos, L.B., Bourne, P.E., et al.: The FAIR guiding principles for scientific data management and stewardship. Scientific Data **3**(1), 1–9 (2016)
27. Wozniak, J.M., Armstrong, T.G., Wilde, M., Katz, D.S., Lusk, E., Foster, I.T.: Swift/T: Scalable data flow programming for distributed-memory task-parallel applications. In: Proc. CCGrid (2013)
28. Wozniak, J.M., Jain, R., Balaprakash, P., Ozik, J., Collier, N., Bauer, J., Xia, F., Brettin, T., Stevens, R., Mohd-Yusof, J., Cardona, C.G., Essen, B.V., Baughman, M.: CANDLE/Supervisor: A workflow framework for machine learning applied to cancer research. BMC Bioinformatics **19**(18), 491 (2018). https://doi.org/10.1186/s12859-018-2508-4, https://doi.org/10.1186/s12859-018-2508-4

29. Wozniak, J.M., et al.: Braid-DB GitHub repository, https://github.com/ANL-Braid/DB

9 Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357.

The following text will be removed in the final submission: