PROV-IO$^+$: A Cross-Platform Provenance Framework for Scientific Data on HPC Systems

Runzhou Han, Mai Zheng, Suren Byna, *Member, IEEE*, Houjun Tang, Bin Dong, Dong Dai, Yong Chen, Dongkyun Kim, Joseph Hassoun, and David Thorsley

***Abstract*—Data provenance, or data lineage, describes the life cycle of data. In scientific workflows on HPC systems, scientists often seek diverse provenance (e.g., origins of data products, usage patterns of datasets). Unfortunately, existing provenance solutions cannot address the challenges due to their incompatible provenance models and/or system implementations. In this paper, we analyze four representative scientific workflows in collaboration with the domain scientists to identify concrete provenance needs. Based on the first-hand analysis, we propose a provenance framework called PROV-IO$^+$, which includes an I/O-centric provenance model for describing scientific data and the associated I/O operations and environments precisely. Moreover, we build a prototype of PROV-IO$^+$ to enable end-to-end provenance support on real HPC systems with little manual effort. The PROV-IO$^+$ framework can support both containerized and non-containerized workflows on different HPC platforms with flexibility in selecting various classes of provenance. Our experiments with realistic workflows show that PROV-IO$^+$ can address the provenance needs of the domain scientists effectively with reasonable performance (e.g., less than 3.5% tracking overhead for most experiments). Moreover, PROV-IO$^+$ outperforms a state-of-the-art system (i.e., ProvLake) in our experiments.**

***Index Terms*—Data provenance, HPC I/O libraries, high performance computing (HPC), scientific data management, workflows.**

I. Introduction

A. Motivation

DATA-DRIVEN scientific discovery has been well acknowledged as a new fourth paradigm of scientific innovation [1]. The shift toward the data-driven paradigm imposes new challenges in data findability, accessibility, interoperability, reusability (i.e., FAIR principles [2], [3]) and trustworthiness [4], all of which demand innovative solutions for modeling and capturing provenance, i.e., the lineage of data life cycle.

As an example, Fig. 1 shows a simplified scientific workflow which analyzes geophysical sensing data on high performance computing (HPC) systems (i.e., DASSA [5]). The workflow takes geophysical data as input, which are often stored in different file formats (e.g., ".tdms", ".h5"). It then converts non-HDF5 files into a uniform HDF5 format (i.e., ".h5"). Depending on the analysis goals, the workflow further applies a set of analysis programs (e.g., "Decimate", "X-Correlation-Stacking") to process the files, the results of which are stored as data products in HDF5 format.

Based on our survey, the domain scientists using DASSA need the fine-grained origin of the data products (i.e., *backward data lineage*). For example, *User A* applies the "Decimate" program with a number of HDF5 files as input and generates a set of data products. Another *User B* may query the origin of the datasets in the final data products to understand which datasets in the input files contributed to which portions of the final data products, or who initiated the "Decimate" application to generate the data products and when. Such provenance information is important for ensuring the reproducibility, explainability, and security of the DASSA data. Nevertheless, the DASSA workflow involves multiple programs accessing multiple files using different I/O interfaces and operations (e.g., HDF5 and POSIX), which makes tracking and deriving the data provenance non-trivial. Moreover, as we will elaborate in Section III, there are other diverse needs of provenance for different scientific workflows and data (e.g., I/O statistics, configuration lineage). Such diversity, complexity, as well as the stringent performance requirement in HPC environments call for a practical solution beyond the state of the art.

B. Limitations of State-of-The-Art Tools

Unfortunately, to the best of our knowledge, existing provenance tools cannot address the grand challenge above sufficiently due to a number of limitations:

First, while the importance of provenance has been well recognized across communities in general (e.g., databases [6], [7], [8], [9], [10], operating systems (OS) [11], [12], eScience [13], [14], [15], [16], [17]), there is a lack of concrete understanding of the exact provenance needs of domain scientists, largely due to the variety of data and metadata that could be generated from HPC systems. As a result, existing solutions are often too coarse-grained (e.g., whole file tracking without understanding HPC data formats [11]) to help domain scientists effectively,

Manuscript received 1 January 2023; revised 24 February 2024; accepted 28 February 2024. Date of publication 14 March 2024; date of current version 29 March 2024. This work was supported in part by NSF under Grant CNS-1855565, Grant CCF-1853714, Grant CCF-1910747, and Grant CNS-1943204 and in part by DOE under Grant DE-AC02-05CH11231. Recommended for acceptance by V. Cardellini. *(Corresponding author: Mai Zheng.)*

Runzhou Han and Mai Zheng are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50014 USA (e-mail: mai@iastate.edu).

Suren Byna is with The Ohio State University, Columbus, OH 43210 USA.

Houjun Tang and Bin Dong are with Lawrence Berkeley National Laboratory, Berkeley, CA 94720 USA.

Dong Dai is with the University of North Carolina, Charlotte, NC 28223 USA.

Yong Chen, Dongkyun Kim, Joseph Hassoun, and David Thorsley are with Samsung Research Labs, Mountain View, CA 94043 USA.

Digital Object Identifier 10.1109/TPDS.2024.3374555

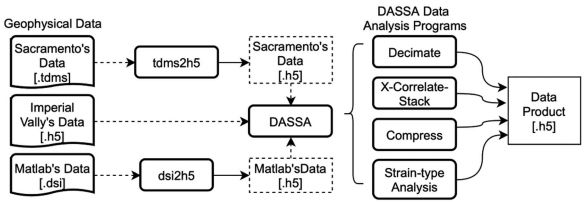


Fig. 1. DASSA workflow. Solid arrows stand for write operation and dashed arrows stand for read operation.

or too specific for one use case (e.g., Machine Learning [18]) to support general needs.

Second, in terms of provenance modeling, we find that existing fundamental standards (e.g., W3C PROV [19]) are not directly applicable to describing the characteristics of scientific data provenance precisely. Scientists often seek a variety of information from scientific workflows on HPC systems, including the origins of data products, the configurations used for deriving results, the usage patterns of datasets, and so on, which cannot be described effectively using any existing provenance models. Additional HPC workflow terms are required to improve the capability of existing provenance solutions for describing scientific data.

Third, in terms of usability, existing approaches often require the users to identify the critical code sites in the workflow software (e.g., loop structure [20]) and manually insert API calls to track the desired information accordingly. Moreover, they often rely on many external packages to work properly, which make it difficult to deploy and use them on different HPC platforms. The labor-intensive and error-prone approaches, together with the portability and compatibility issues, hinder the wide adoption of provenance products and diminishes the potential benefits.

Note that the limitations highlighted above are correlated. For example, the lack of understanding of provenance needs and the ambiguity of the provenance model are contributing to each other, which fundamentally limits the usability of existing solutions in terms of granularity, expressibility, etc., which in turn makes clarifying the ambiguity and real needs difficult.

C. Key Insights & Contributions

We tackle the grand challenge of provenance support for scientific data on HPC systems in this paper.

First, we observe that for a provenance framework to be practical and useful, inputs from the end users (i.e., domain scientists) is essential. Therefore, we collaborate with domain scientists to analyze four representative scientific workflows in depth.

In doing so, we identify the unique characteristics of the workflows studied (e.g., I/O interfaces, data formats, access patterns) as well as the specific needs for scientific data provenance (e.g., lineage at file, dataset, or attribute granularity).

Second, we observe that I/O operations are critically important in affecting the state of data that form the lineage needed by the domain scientists.

Therefore, different from existing solutions [20], [21], [22], we introduce an I/O-centric provenance model dedicated for the HPC environments. The model is derived from the W3C PROV standard [19] with a variety of concrete sub-classes, which can describe both the data and the associated I/O operations and execution environments precisely with extensibility. Moreover, it enables us to decouple the data provenance from specific executions of a workflow and support the integration of provenance from multiple runs naturally, which is important as workflows may evolve over time.

Third, based on the fine-grained provenance model, we find that the rich I/O middleware already used by the scientists provide an ideal vehicle for capturing the desired provenance transparently. Therefore, we create a configurable and extensible library and integrate it with existing I/O code paths (e.g., HDF5 I/O and POSIX syscalls) to capture necessary information without requiring the scientists to modify the source code of their workflows.

Moreover, to further improve the usability, we persist the captured provenance as standard RDF triples [23] and enable provenance query and visualization.

Forth, through the communication with domain scientists in the industry, we notice the increasing importance of supporting containerization [24]. By wrapping the HPC workflows together with their dependencies in containers, the containerization techniques can effectively reduce the burden of software maintenance and thus enable more desired features including reproducibility, reusability, interoperability, etc. Therefore, a provenance framework should be generic enough to handle provenance in both containerized and non-containerized scenarios.

Based on the key ideas above, we build a framework called PROV-IO+, which can provide end-to-end provenance support for domain scientists with little manual effort across HPC platforms. We deploy PROV-IO+ on representative supercomputers and evaluate it with realistic workflows. Our experiments show that PROV-IO+ incurs reasonable performance overhead and outperforms a state-of-the-art provenance prototype (i.e., IBM ProvLake [20]) for the use cases evaluated. More importantly, through the query and visualization support, PROV-IO+ can address the provenance needs of the scientists effectively.

In summary, we have made the following contributions:

- Identifying concrete provenance needs of domain scientists based on four representative scientific workflows;
- Designing a comprehensive PROV-IO+ model to describe the provenance of scientific data precisely and extensibly;
- Building a practical prototype of PROV-IO+ which can support different HPC workflows with little human efforts in both containerized and non-containerized scenarios;
- Measuring the PROV-IO+ prototype in HPC environments and demonstrating the efficiency and effectiveness;
- Releasing PROV-IO+ as an open-source tool to facilitate follow-up research on provenance in general.

D. Experimental Methodology & Artifact Availability

Experiments were performed on three state-of-the art platforms, including LBNL Cori Supercomputer [25], Samsung SAIT Supercomputer (S-SuperCom) [26], and Google Cloud Platform (GCP) [27]. First, in terms of non-containerized scenario, we applied PROV-IO+ to three scientific workflows (i.e., DASSA [5], Top Reco [28], and an I/O-intensive application

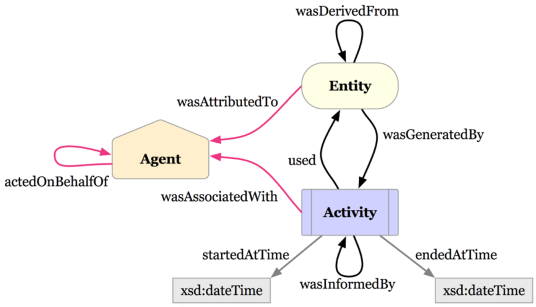


Fig. 2. W3C Provenance Model [31].

based on H5bench [29]) on the Cori supercomputer. Second, in terms of containerized scenario, we applied PROV-IO$^+$ to analyze one deep learning workflow (i.e., Megatron-LM [30]) on S-SuperCom. As will be discussed in Section III, these use cases cover diverse characteristics (e.g., various languages, file formats, I/O interfaces, metadata) and provenance needs (e.g., file/dataset/attribute lineage, metadata versioning, I/O statistics).

We varied the critical parameters of the workflows to measure the run-time performance and storage requirements under a wide range of scenarios. Third, we examined container's impact on provenance tracking with Megatron-LM [30] on the Google Cloud Platform where we can schedule both containerized and non-containerized workflows and perform a fair comparision. In addition, we compared PROV-IO$^+$ with ProvLake [20] using the Python-based Top Reco workflow as ProvLake only supports Python at the time of this writing. The PROV-IO$^+$ prototype is open-source at https://github.com/hpc-io/prov-io.

II. Background

A. W3C Provenance Standard

The PROV family of specifications, published by the World Wide Web Consortium (W3C), is a set of provenance standard to promote provenance publication on the Web with interoperability across diverse provenance management systems [31]. One key specification is PROV-DM, an extensible relational model which describes provenance information with a graph representation. As shown in Fig. 2, a W3C provenance graph abstracts information into classes of *Entity*, *Activity* and *Agent*. PROV-DM also defines *Relation* among the three classes. Another critical specification is PROV-O which describes the mapping of PROV-DM classes to RDF triples. In PROV-O, *Entity*, *Activity* and *Agent* are mapped to subjects and objects, while *Relation* is mapped to predicates. We inherit these key notions from W3C PROV standard in the PROV-IO$^+$ design.

B. HPC I/O Libraries

I/O libraries (e.g., ADIOS [32], HDF5 [33], and NetCDF [34]) play an essential role in scientific computations. Many workflows leverage the library I/O to manipulate data files. For example, HDF5 (i.e., Hierarchical Data Format version 5) is one of the the most wildly used I/O libraries for scientific data [35]. It is developed to be a parallel data management middleware to bridge the gap between HPC applications and the complicated, low-level details of underlying file systems, and has grown to a popular data format and management system.

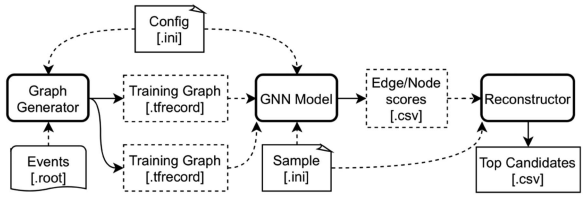


Fig. 3. Top Reco workflow. Solid arrows stand for write operation and dashed arrows stand for read operation.

In this work, we integrate our solution with the HDF5 library besides the classic POSIX I/O operations. This is based on the observation that HDF5 has evolved with a Virtual Object Layer (VOL) which can intercepts object-level API operations to functional plugins, called VOL Connectors [36]. VOL connectors allow third-party developers to add desired storage functionalities, which can be loaded dynamically at runtime. We leverage such extensibility for tracking the provenance of HDF5 I/O data.

III. Case Studies

In this section, we discuss four real-world use cases to motivate the I/O-centric provenance further. For each case, we describe its semantics and characteristics, the provenance need of the domain scientists, and the associated challenges.

A. Top Reco - Lineage of Configurations

Workflow Description: Top Reco [28] is a Machine Learning (ML) workflow in high-energy physics data analysis, which uses Graph Neural Network (GNN) models for top quark reconstruction. Top quarks are the elementary particles with the most mass that may decay quickly and are not detectable directly due to their mass. By representing particles and their relationships as graphs, the GNN-based workflow can help reconstruct top quarks more accurately and efficiently, which is important for physics discoveries.

In Fig. 3, we show the key steps of the Top Reco workflow. First, the workflow takes two types of files as input, including the ".root" file for input event and the ".ini" file for configuration. Second, it generates ".tfrecord" files which stores the training dataset and test dataset based on the input events. Third, it trains a GNN model with the training dataset and tests the model with the test dataset by accessing the ".tfrecord" files. Fourth, a range of scores of edge and nodes are generated as the output of the model. Finally, a reconstructor component runs a simulation of reconstructing the top quarks based on highest scores. As summarized in Table I, the Top Reco workflow uses the POSIX I/O interface, and involves multiple programs accessing multiple files.

Provenance Need: In the Top Reco case, the domain scientists are interested in the impact of GNN configurations on the model performance. Specifically, they would like to know which combination of model hyperparameters and dataset preselections result in the best training accuracy. In other words, they would like to have fine-grained version control of the metadata (e.g., hyperparameters, preselections) as well as the correlation between the metadata and the result to ensure the explainability and reproducibility.

Challenges: Essentially, the Top Reco case requires automatic version control management on the machine learning model. However, a typical version control system (e.g., Git) cannot

TABLE I
FOUR REAL USE CASES WITH DIFFERENT CHARACTERISTICS AND PROVENANCE NEEDS

Use Case	Description	I/O Interface	Provenance Need
Top Reco	training GNN models for top quark reconstruction; multi-program, multi-file;	POSIX	metadata version control & mapping
DASSA	parallel processing of acoustic sensing data; multi-program, multi-file;	HDF5 & POSIX	backward lineage of data products
H5bench	simulating typical I/O patterns of HDF5 app; multi-program, single-file;	HDF5	I/O statistics & bottleneck
Megatron-LM	parallel transformer model for NLP; multi-program, multi-file;	POSIX	Checkpoint-configuration consistency

meet the requirements because it cannot automatically track the model performance and maps the performance to the model configuration. In practice, the scientists may need to execute the workflow for multiple times with different configurations, and each execution may take multiple hours or more. Due to lack of provenance support, the scientists have to manually make a new copy of configuration when they start a new run, and record the corresponding result later. Such common practice is time-consuming and not scalable. In other words, a new provenance framework is urgently needed.

B. DASSA - Lineage of Data Products

Workflow Description: As mentioned in Section I-A, DASSA [5] is a parallel storage and analysis framework for distributed acoustic sensing (DAS). It uses a hybrid (i.e., MPI and OpenMP) data analysis execution engine to support efficient and automated parallel processing of geophysical data in HPC environments, which has been applied for accelerating a variety of scientific computations including earthquake detection, environmental characterization, and so on. The overall workflow is described in Fig. 1.

Provenance Need: As discussed in Section I-A, the domain scientists need the *backward data lineage* to understand the origin of the data products and to ensure the data reproducibility, explainability, and security, among others.

Challenges: The DASSA workflow may involve multiple different programs, file formats, I/O interfaces, and end users, which is representative for large-scale scientific workflows in HPC environments. Moreover, both the file level and the sub-file level (e.g., inner hierarchies of the HDF5 format) information is needed. To the best of our knowledge, none of the existing provenance models or systems can handle the complexity to meet the comprehensive needs.

C. H5bench - Data Usage and I/O Performance

Workflow Description: H5bench [29] is a parallel I/O benchmark suite for HDF5 [37] that is representative of various large-scale workflows. It includes a default set of read and write workloads with typical I/O patterns in HDF5 applications on HPC systems, which enables creating synthetic workflows to simulate diverse HDF5 I/O operations in HPC environments. The benchmark also contains 'overwrite' and 'append' operations that allow modifying data or metadata of existing files and appending new data, respectively.

We collect an H5bench-based workflow which contains a combination of 'write', 'overwrite', 'append' and 'read' workloads operating on HDF5 files via MPI. This workflow simulates the typical scenarios where a single file may be accessed concurrently by HPC applications and multiple versions of a dataset may be generated accordingly. As shown in Table I, the H5bench-based workflow mainly uses the HDF5 I/O interface, and involves multiple programs accessing a single file.

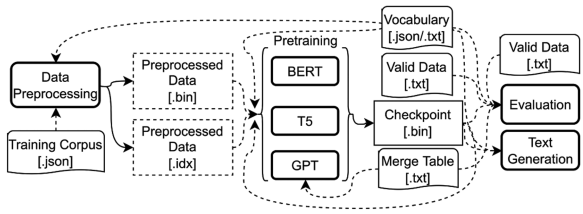


Fig. 4. Megatron-LM Workflow. Solid arrows stand for write operation and dashed arrows stand for read operation.

Provenance Need: Understanding frequently accessed data in large datasets leads to optimizing I/O performance by improved data placement and layout. Scientists typically use the H5bench-based workflow to collect I/O statistics and identify potential bottlenecks on HPC systems. While I/O profiling tools, such as Darshan [38] and Recorder [39] collect coarse-grained statistics of I/O performance, there are no tools to extract data access information and the cost of those operations. Fine-grained information such as the total number of each type of HDF5 I/O operations incurred during the workflow, the accumulated time cost for each type of operations, the distribution of operations and time overhead, the HDF5 APIs invoked at a specific time point, etc. would be critically important for understanding the system behavior and fine-tuning the performance.

Challenges: The H5bench use case involves handling HDF5 datasets concurrently and measuring diverse fine-grained metrics at the HDF5 API level, which requires deep understanding of the semantics and internals of HDF5. Since existing solutions are largely incompatible with HDF5, they are fundamentally inapplicable for this important category of use cases.

D. Megatron-LM - Checkpoint Consistency

Workflow Description: Megatron-LM is a powerful transformer [40] (i.e., a type of deep learning models) developed by NVIDIA [30], [41]. Megatron-LM supports training large transformer language models at scale, which is achieved by providing efficient, model-parallel (tensor, sequence, and pipeline), and multi-node pre-training of transformer-based models (e.g., GPT [42], BERT [43], and T5 [44]) using mixed precision. Megatron-LM scales the transformer training by supporting data parallelism and model parallelism further. Specifically, the data parallelism is achieved by splitting the input dataset across specified devices (e.g., GPUs); on the other hand, the model parallelism is implemented by splitting the execution of a single transformer module over multiple GPUs working on the same dataset. Both data parallelism and model parallelism features can be optionally configured in the workflow, and they are both enabled in this study for completeness. Fig. 4 shows a simplified overview of the Megatron-LM workflow. First, a training corpus (".json") is preprocessed by the data processing module, which generates a binary file (".bin") and an index file (".idx"). The preprocessed data become the input of the pre-training transformer models. A trained model (i.e., checkpoint)

will be generated at the end of pretraining, and it can be used in the follow-up evaluation or text generation. Users may also skip the pretraining step if they already have a trained model available.

Provenance Need: In the Megatron-LM workflow, the consistency between pretraining models's checkpoint and the corresponding configuration is important. The checkpoint mainly records the metadata of the previous pretraining process, such as micro/global batch size and state of optimizer/scheduler, which are dependent on the configuration parameters. Blindly modifying configuration parameters in the next pretraining could easily result in various types of errors (e.g., network errors, test code errors). Moreover, many configuration parameters are tightly correlated with each other, changing configuration parameters without preserving the correlation may also lead to failures. In addition, adjusting pretraining hyperparameters incautiously may affect the model quality negatively (e.g., result in overfitting). Therefore, in the Megatron-LM use case, the domain scientists want to track the checkpoint and configuration provenance to ensure the checkpoint-configuration consistency.

Challenges: The challenge for the Megatron-LM workflow is two-fold. First of all, the workflow involves hundreds of configuration entries and checkpointed statuses which are difficult to track or reason by human.

Due to the lack of tool support, the domain scientists cannot manage checkpoints generated by multiple training processes conveniently and identify a qualified checkpoint and the associated configurations consistent with new training processes.

Moreover, with the growing popularity of container technologies [45], [46], HPC systems have started to be integrated with container-based job runtime tools [24]. In such HPC systems, large-scaled parallel pretraining models are executed in a containerized environment, which largely avoids tedious efforts in resolving installation or runtime dependencies (e.g, PyTorch [47] and nccl [48]). Besides the challenge of the Megatron-LM workflow itself, the domain scientists using the workflow would like to execute the workflow in a containerized HPC environment. Since a container is an isolated environment by design where application cannot directly interactive with the host system, how to containerized Megatron-LM and track provenance information of the containerized workflow at scale on HPC systems is another major challenge in this use case.

Note that both Megatron-LM and Top Reco (Section III-A) belong to deep learning provenance use cases. However, Megatron-LM has two major differences compared to Top Reco. First, in Megatron-LM, the provenance need is checkpoint-configuration consistency, while Top Reco's provenance need is configuration version control. Second, Top Reco is a traditional single thread workflow, while Megatron-LM is a containerized parallel workflow, which introduces more challenges in terms of both provenance tracking and provenance storage.

E. Summary

By analyzing the four cases in depth and consulting with the domain scientists, we find that there is a big gap between the provenance needs and existing solutions. The variety of the workflow characteristics (e.g., different I/O interfaces and file formats) as well as the diversity of scientists' needs motivates us to design a comprehensive provenance framework to address the challenge, which we elaborate in the following sections.

IV. PROV-IO$^+$ Design

In this section, we introduce the design of PROV-IO$^+$. We focus on the provenance model (Section IV-A) and its system architecture (Section IV-B), which are two fundamental pillars of PROV-IO$^+$. We defer additional implementation details to the next section (Section V).

A. PROV-IO$^+$ Model

Fig. 5(a) shows an overview of the PROV-IO$^+$ model, which is derived based on the W3C standard (Section II-A) as well as the characteristics of typical workflows and the provenance needs of domain scientists (Section III).

Inspired by the W3C specification, we classify information into four PROV-IO$^+$ super-classes: *Entity* (yellow boxes in Fig. 5(a), *Activity* (purple), *Agent* (orange) and *Extensible Class* (green). Moreover, we introduce a variety of concrete sub-classes to enrich the model, which can capture the data with different granularity as well as the associated I/O operations and execution environments for deriving the data. New *Relation* terms among the sub-classes are also introduced (text on arrows). Note that our goal is to design a model suitable for our HPC use cases instead of strictly following existing standards, and we do not claim our model is strictly compatible with existing models due to our new additions. We leave the full compatibility as future work. We summarize the definitions of the sub-classes in Table II and highlight the main concepts added to each super-class/relation as follows:

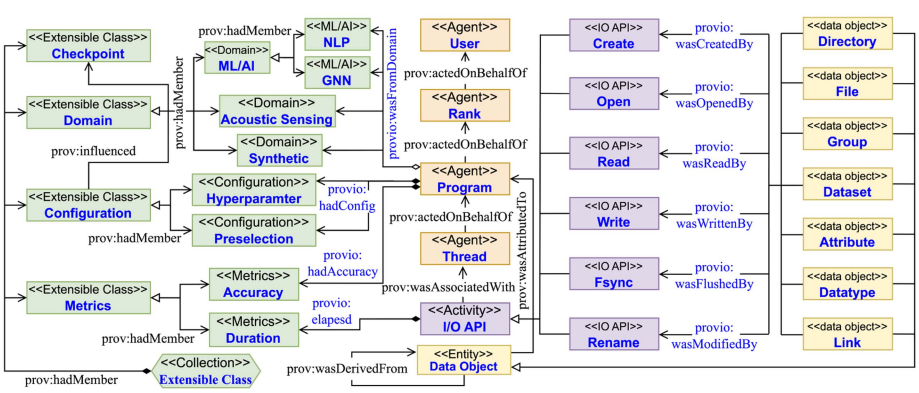
1) Entity: This PROV-IO$^+$ super-class includes seven specific *Data Object* sub-classes/types (i.e., *Directory*, *File*, *Group*, *Dataset*, *Attribute*, *Datatype*, *Link*). Together, these sub-classes cover common I/O structures and file formats. For example, *Attribute* is a combined sub-class that can map to both the HDF5 attributes and the extended attributes of an inode in a POSIX-compliant Ext4 file system [49].

2) Activity: This super-class includes six specific *I/O API* sub-classes /types (i.e., *Create*, *Open*, *Read*, *Write*, *Fsync*, *Rename*). These sub-classes cover a wide range of commonly used I/O operations in HPC environments. For example, *Read* can map to HDF5 read-family operations (e.g., "H5Gread", "H5Dread", "H5Aread", "H5Tread") and POSIX system call "read" and its variants. Note that these operations are applicable to other I/O libraries too (e.g., NetCDF [34]).

3) Agent: This super-class includes a set of sub-classes representing the operator of a series of activities, such as *Thread*, *User*, *Rank*, and *Program*. This fine-grained representation is necessary because HPC applications are typically multi-threaded and are executed in parallel (e.g., a group of MPI processes with different ranks running on a cluter of nodes).

4) Extensible Class: This super-class contains a hierarchical collection of properties pertained by entities, activities and agents. It is designed to be extensible because valuable information is often workflow-specific. In the prototype, we define four generic sub-classes (i.e., *Checkpoint*, *Type*, *Configuration*, *Metrics*) to cover a variety of valuable information that cannot be described precisely in the native W3C (e.g., hyperparameters of ML models, checkpoints of AI model training).

5) Relation: PROV-IO$^+$ Relation describes the diverse relations among classes. We inherit the basic W3C provenance relations between *entity* & *entity* (prov:wasDerivedFrom), *entity* & *agent* (prov:wasAttributedTo), *activity* & *agent*



(a) PROV-IO+ model. (b) A Provenance Snippet.

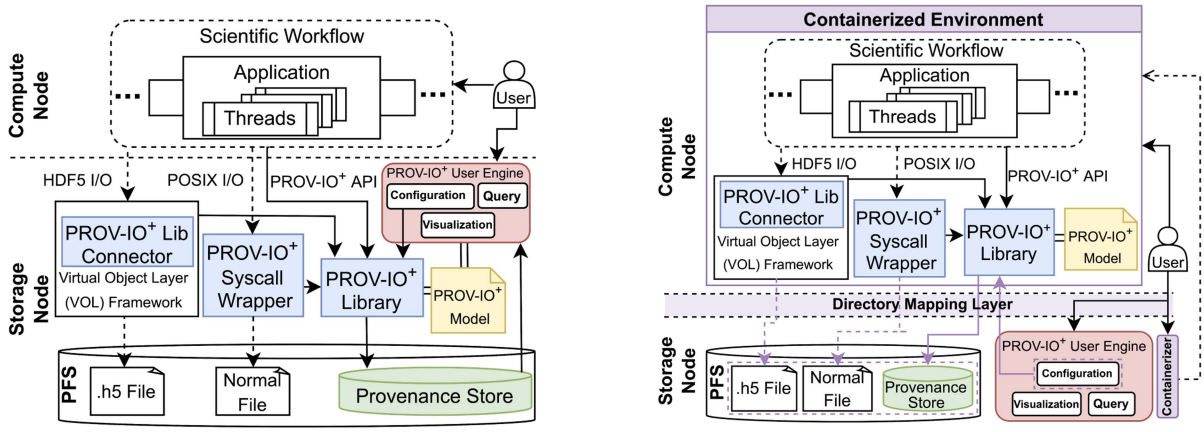
Fig. 5. PROV-IO+ Model Overview. (a) The PROV-IO+ model classifies information into four super-classes: *Entity* (yellow boxes), *Activity* (purple boxes), *Agent* (orange boxes) and *Extensible Class* (green boxes). The new concepts introduced by PROV-IO+ are highlighted with blue color. (b) A Provenance Snippet based on PROV-IO+.

TABLE II
DESCRIPTION OF PROV-IO+ MODEL

Super-class	Sub-class	Description
Entity	<<Data Object>> Directory	Stands for directories (inode) in file system.
	<<Data Object>> File	Stands for individual file (inode) in file system.
	<<Data Object>> Group	Stands for HDF5 group. Groups contain datasets, attributes and other HDF5 structures.
	<<Data Object>> Dataset	Stands for HDF5 dataset. Dataset contains scientific data.
	<<Data Object>> Attribute	A combined class with semantic of both file system and HDF5. It maps to both HDF5 attribute and inode extended attribute.
	<<Data Object>> Datatype	Stands for HDF5 datatype. To be noticed HDF5 datatype is different to "xsd:datatype".
	<<Data Object>> Link	Stands for hard/soft links in file system. HDF5 also has a link structure but we don't consider it in this work.
Activity	<<I/O API>> Create	Stands for create operation in HDF5, including "H5Gcreate" (create group), "H5Dcreate" (create dataset), "H5Acreate" (create attribute) and "H5Tcreate" (create datatype).
	<<I/O API>> Open	Open operation in HDF5, including "H5Gopen" (open group), "H5Dopen" (open dataset), "H5Aopen" (open attribute) and "H5Topen" (open datatype). Also stands for POSIX syscall "open".
	<<I/O API>> Read	Read operation in HDF5, including "H5Gread" (read group), "H5Dread" (read dataset), "H5Aread" (read attribute) and "H5Tread" (read datatype). Also stands for POSIX syscall "read" and its variants.
	<<I/O API>> Write	Write operation in HDF5, including "H5Gwrite" (write group), "H5Dwrite" (write dataset), "H5Awrite" (write attribute) and "H5Twrite" (write datatype). Also stands for POSIX syscall "write" and its variants.
	<<I/O API>> Fsync	Flush operation in both HDF5 and POSIX syscall interface, including H5Flush and POSIX syscall "fsync" and its variants.
	<<I/O API>> Rename	POSIX syscall "rename" and its variants.
Agent	User	Stands for an individual user.
	Thread	Thread/Process info. in multi-threaded programs with concurrent I/O (e.g., MPI, OpenMP, PThreads).
	Program	A program instance. One program may have multiple running instances on different MPI ranks. If a program is single-threaded, the program is directly "actedOnBehalfOf" a user.
Extensible Class	Domain	The research domain of a program/workflow. Currently it includes ML/AI, Acoustic Sensing, and Synthetic.
	Configuration	Program/model/workflow configurations. E.g., the GNN hyperparameter and data pre-selection in the Top Reco workflow.
	Metrics	Evaluation metrics of the workflow. In H5bench workflow, to analyze performance bottleneck, I/O API duration is a primary metric. In Top Reco and other ML workflows, model accuracy is a common metrics to be tracked. Yet PROV-IO+ doesn't include other ML metrics (e.g., precision, recall, F-score, etc) in our basic version of PROV-IO+.
	Checkpoint	Program/model/workflow checkpoints. E.g., the pretraining checkpoint of Megatron-LM.

(prov:AssociatedWith), *agent* & *agent* (prov:actedOnBehalfOf). Moreover, we introduce new relations between *entity* & *activity* to precisely describe the relations between various I/O API and Data Object sub-classes (e.g., provio:wasCreatedBy, provio:wasReadBy, provio:wasWrittenBy, provio:wasModifiedBy). PROV-IO+ also defines new relations between original classes (i.e., Agent, Activity) and Extensible class (e.g., provio:wasFromDomain, provio:hadConfig).

To make the description more concrete, we show an example snippet of provenance captured by PROV-IO+ in Fig. 5(b), which contains eight records pertained by different subjects.



(a) Mode #1: Provenance for Non-Containerized Workflow.

(b) Mode #2: Provenance for Containerized Workflow.

Fig. 6. Architecture of PROV-IO$^+$ Framework. There are two usage modes: (a) Mode #1 provides provenance support for non-containerized workflows; (b) Mode #2 supports containerized workflows. The framework includes five major components in total: the PROV-IO$^+$ model (yellow), a provenance tracking engine (blue modules), a provenance store (green), a user engine (red), and a containerizer engine (purple).

Each subject can be a sub-class of an *Agent* (e.g., "Bob", "vpicio_uni_h5–a1.exe"), an *Activity* (e.g., "H5Dcreate2–b1"), or an *Entity* (e.g., "/Timestep_0/x"). Each record is a series of triples starting with a unique subject, where the triples describe provenance information of a subject. Note that the record length may vary depending on the provenance information associated with the subject. Given this snippet, we can derive complex provenance information (e.g., dataset "/Timestep_0/x" was created by I/O API "H5Dcreate2–b1" associated with program "vpicio_un_h5.exe–a1", which was started by user "Bob").

B. PROV-IO$^+$ Architecture

Fig. 6 shows the architecture of the PROV-IO$^+$ framework, which supports two usage modes: Mode #1 (Fig. 6(a)) provides provenance support for traditional non-containerized workflows on HPC systems; Mode #2 (Fig. 6(b)) supports containerized workflows. There are five components in total, including: (1) the PROV-IO$^+$ model (yellow) to specify the provenance information Section IV-A; (2) a provenance tracking engine (blue modules) which captures I/O operations from multiple I/O interfaces; (3) a provenance store (green) which persists captured provenance into RDF triples; (4) a user engine (red) for users to query and visualize provenance; (5) a containerizer engine (purple) to support other components in containerized environments. Among the five components, the PROV-IO$^+$ model (yellow) has been discussed in details in Section IV-A. We introduce the other three common components used in both modes (i.e., provenance tracking, provenance store, and user engine) one by one in Section IV-B1, and then discuss the containerizer engine for supporting containerized workflows in Section IV-B2.

1) Mode #1: Support for Classic Workflows: To provide provenance support for the classic, non-containerized workflows (i.e., Mode #1), PROV-IO$^+$ leverages three major components based on its provenance model (Section IV-A) as follows:

Provenance Tracking: As shown in Fig. 6(a), a scientific workflow is typically started on compute nodes. The workflow may consist of several parallel applications with multiple threads running concurrently. During the workflow execution, all I/O operations (e.g., POSIX and HDF5) are monitored by PROV-IO$^+$ for provenance collection.

Specifically, the Provenance Tracking component contains two thin modules (i.e., *PROV-IO$^+$ Lib Connector* and *PROV-IO$^+$ Syscall Wrapper*) for monitoring the library I/O and POSIX I/O operations respectively. In case of the HDF5 library, the PROV-IO$^+$ Lib Connector (we use the alias LibConnector in the remainder of the article) monitors the I/O requests within the HDF5 Virtual Object Layer (VOL). In case of POSIX, the I/O syscalls are monitored through the PROV-IO$^+$ Syscall Wrapper which is configurable via environmental variables. In both cases, PROV-IO$^+$ let the native I/O requests pass through and invoke the core *PROV-IO$^+$ Library* for collecting the provenance defined by the PROV-IO$^+$ model without changing the original I/O semantics.

Note that both the library I/O and POSIX I/O operations can be tracked in a transparent and non-intrusive way from the workflow's perspective, which is important for usability.

In addition, to achieve extensibility, we provide a set of PROV-IO$^+$ APIs which enables users to convey user/workflow-specific semantics and requirements to PROV-IO$^+$ (i.e., Extensible Class in PROV-IO$^+$ model). Similar to ProvLake [20], users can instrument their workflows with PROV-IO$^+$ APIs as needed (e.g., tracking a specific hyperparameter of a ML workflow). By providing such flexibility, additional provenance needs can be satisfied by PROV-IO$^+$ conveniently.

Provenance Store: The Provenance Store component maintains the provenance information as RDF graphs durably on the underlying parallel file system to enable future queries. We choose an RDF triplestore instead of a traditional SQL database for two main reasons: (1) W3C PROV-DM already has a well-defined ontology (i.e., PROV-O [19]) to map the model to RDF, so using RDF makes PROV-IO$^+$ compatible with other W3C-compliant solutions; (2) To answer path queries in provenance use cases, SQL queries with repeated self-joins are necessary to compute the transitive closure, which often leads to worse performance when the provenance grows [50].

More specifically, the Provenance Store component provides an interface for the PROV-IO$^+$ Library to manipulate provenance records and maintain provenance graphs efficiently, which includes creating a new provenance RDF graph in memory, loading an existing graph, inserting new records to an existing graph, etc. To minimize the performance impact on the workflow, the in-memory provenance graph is serialized to the Provenance

Store asynchronously. And depending on the need of the user, the serialization operation may be triggered either periodically or by the end of the workflow.

PROV-IO+ User Engine: The provenance information could be enormous due to the complexity of scientific workflows. To avoid distraction and help users derive insights, the PROV-IO+ User Engine component allows users to enable/disable individual sub-classes defined in the PROV-IO+ model, which also enables flexible tradeoffs between completeness and overhead.

Moreover, the engine provides a query interface to allow the user to issue queries on the provenance generated by PROV-IO+. Moreover, it includes a visualization module to visualize the provenance (sub)graphs requested by the user. Note that both the query and the visualization need to follow the PROV-IO+ model, which enforces a uniform way to represent the rich provenance information.

Note that in the preliminary version of the prototype [51], the user engine only provides a basic query interface to users. As a result, users have to issue query primitives one by one to achieve a complicated provenance query. In the current prototype, PROV-IO+ is further equipped with a set of high-level integrated query APIs for answering typical provenance needs, which can simplify the query complexity and improve the usability for end users further. For example, in the DASSA use case, to track the backward lineage of an output file, an user only needs to provide the name of the file and the level of predecessor through the integrated query APIs, which will retrieve the provenance information conveniently.

2) Mode #2: Support for Containerized Workflows: To provide provenance support for containerized workflows, PROV-IO+ includes an additional component called *containerizer* besides the components discussed above.

The containerizer engine provides two main functionalities. First, it assembles the target workflows (including their dependencies) as well as the PROV-IO+ common modules (Section IV-B1) into container images to be executed on container platforms. For example, on an HPC system using Singularity/Apptainer [24] (we call it HPC container for short in the rest of the article), the containerizer engine first creates the Docker [45] image for the workflow and then converts the that image into an HPC container image for execution on compute nodes.

Second, the containerizer engine establishes the mapping between the directory namespace within the container and the namespace outside the container on the HPC storage nodes, and re-directs the relevant provenance I/O activities to the provenance store for persistency, as shown in the "Directory Mapping Layer" and the purple dash lines in Fig. 6(b). In this way, PROV-IO+ can support containerized workflows on HPC systems automatically with little additional efforts. More implementation details will be discussed in the next section.

V. PROV-IO+ Implementation

In this section, we discuss additional implementation details of the major components in the PROV-IO+ framework.

Provenance Tracking: To support HDF5 I/O, we implement the LibConnector in C language and integrate it with the native HDF5 VOL-provenance connector (we use the alias H5Connector in the remainder of the article), which follows a homomorphic design in which each HDF5 native I/O API has a counterpart API [36]. Upon each invocation of an HDF5 native API, the counterpart API adds the corresponding virtual data object to a linked list. LibConnector leverages the linked list with locking support to achieve concurrency control on I/O operations on the same data object. To collect provenance, the PROV-IO+ Library APIs are invoked. We collect *Agent* information at the initialization stage of the native H5Connector. *Entity* and *Activity* classes are tracked at each homomorphic API during the workflow runtime.

Similarly, to support POSIX I/O, we use GOTCHA [52] to build a C wrapper layer for POSIX syscall and invokes the PROV-IO+ Library internally. Additionally, the current PROV-IO+ APIs support invoking the PROV-IO+ Library from workflows written in multiple languages including Python, C/C++, and Java.

Moreover, to support large-scale ML/AI workflows, we further instrument PyTorch [47], one of the most popular machine learning framework, with the PROV-IO+ library. This enables PROV-IO+ to provide more transparency in supporting ML/AI workflows by capturing specific provenance information needed by this category of workflows (e.g., checkpointing information).

Provenance Store: The Provenance Store is implemented based on Redland `librdf` [53] to serve as the durable backend of the PROV-IO+ Library. We choose Redland because based on our experiences, many other existing RDF solutions are not directly usable in our HPC environments due to compatibility issues in dependent packages and/or operating system (OS) kernels [54], [55], [56], [57], [58].

We utilize Redland's in-memory graph representation and its support for serializing in-memory graph to multiple on-disk RDF formats (e.g., Turtle [59], ntriples [60], etc.). Redland `librdf` also supports the integration of multiple databases as the storage backend (e.g., BerkeleyDB, MySQL, SQLite). In the current prototype, we store provenance information in the Turtle format directly for simplicity.

To avoid potential data races when serializing from multiple processes to the Provenance Store, PROV-IO+ maintains an in-memory sub-graph for each process and lets the process serialize its own sub-graph to a unique RDF file on disk. The sub-graph files are then parsed and merged into a complete provenance graph. Since every node in the graph has a globally unique ID (GUID), merging the sub-graphs does not cause unnecessary duplication. Note that this strategy also helps performance because no extra inter-process communication or synchronization is needed during workflow execution, and the merging can be performed after workflow execution. The offline merging takes between 5 minutes to 1 h based on the number of MPI ranks and amount of information tracked in the workflow. We consider online graph merging as a future work.

PROV-IO+ User Engine: The user engine supports querying RDF triples with SPARQL, which is a semantic query language to retrieve and manipulate data stored in RDF [61]. We use Python scripts as the SPARQL endpoint. A Python interface is provided for users to invoke compound SPARQL query statements with a simple Python function given necessary parameters (e.g., a user can call query function `get_predecessor(data_object, predecessor_level)` to retrieve the predecessor of a data object at a specific level). Note that depending on different use case scenarios, the query can vary a lot, as will be demonstrated in Section VI-G. Note that PROV-IO+ provides highly integrated query APIs for scientists to conveniently retrieve provenance based on their needs. For instance, in DASSA workflow,

to query the backward data lineage, the scientists only need to specify the output data object and the level of its predecessor (i.e., how many steps back) to the query API, and the user engine will return the target information if applicable. Similarly, highly integrated and customized query APIs are developed for remaining workflows based on their provenance use cases. In the current prototype, we utilize Graphviz [62] for RDF graph visualization.

Containerizer Engine: The Containerizer Engine is implemented as a set of scripts to enable the provenance support in the containerized environment conveniently. For example, to support containrized Megatron-LM workflow on the HPC container platform, the Containerizer Engine first creates a Docker image by using the NGC's PyTorch 21.07 as the parent image. Besides the Megatron-LM workflow itself, the image also contains the PROV-IO$^+$ library and related dependencies. After the Docker image is created, it is further converted to a HPC container image in order to run it in the containerized HPC environment. Moreover, the directory namespace in the container image is mapped to the PROV-IO$^+$ provenance store on the storage nodes for for data persistence.

Also, since HPC container provides three running modes (i.e., "run", "exec" and "shell") for different execution scenarios (e.g., interactive jobs and batch jobs), the Containerizer Engine includes different sets of scripts to support different modes. For example, to support running containerized workflows in the batch mode with the IBM Spectrum LSF [63] job scheduler, the Containerizer Engine includes scripts to ensure that the configuration parameters of the PROV-IO$^+$ supported containers are consistent with the LSF batch scripts.

VI. Evaluation

In this section, we evaluate the prototype of the PROV-IO$^+$ framework in representative HPC environments.

First of all, we introduce the experimental methodology and HPC platforms for non-containerized and containerized workflows, respectively (Section VI-A). Next, we evaluate PROV-IO$^+$ with three non-containerized workflows (i.e., Top Reco, DASSA and H5Bench) from two perspectives including the tracking performance (Section VI-B) and the storage requirement (Section VI-C). Similarly, we evaluate PROV-IO$^+$ with one containerized workflow (i.e., Megatron-LM) and measure both the tracking performance and the storage requirement (Section VI-D). Moreover, we analyze the impact of containerization on provenance tracking by comparing the tracking overhead in two versions (i.e., with and without containerization) of Megatron-LM (Section VI-E).

In addition, we compare PROV-IO$^+$ with a state-of-the-art provenance prototype (i.e., ProvLake [20]) (Section VI-F), and evaluate the query effectiveness of PRVO-IO$^+$ for all workflows from the end user's perspective (Section VI-G).

Overall, our experimental results shows that PROV-IO$^+$ can support both non-containerized and containerized workflows effectively. Its tracking overhead is less than 3.5% in more than 95% of our experiments, and it outperforms ProvLake in terms of both tracking and storage overhead.

A. Experimental Methodology

Non-Containerized Workflows: We have evaluated the PROV-IO$^+$ framework for non-containerized workflows on a state-of-the-art supercomputer named Cori, which is a Cray XC40 supercomputer deployed at the National Energy Research Scientific Computing Center (NERSC) with a peak performance of about 30 petaflops. As shown in Table III, Cori uses the Slurm job scheduler and do not use container runtime by default. We conduct experiments on Cori using 64 Intel Xeon "Haswell" processor nodes and up to 4096 cores, unless otherwise specified. The storage backend is a Lustre parallel file system (PFS) with stripe count of 128 and stripe size of 16 MB.

TABLE III
Major Experimental Platforms

	Cori	SAIT SuperCom
Processor	Intel Xeon Phi	AMD EPYC
Cores	622,336	204,160
OS	Cray Linux	Redhat 8
PFS	Lustre	Lustre
Scheduler	Slurm	LSF
Container Runtime	–	Singularity/Apptainer

We apply PROV-IO$^+$ to three representative non-containerized workflows including Top Reco [28], DASSA [5], and an H5bench-based workflow [29]. As mentioned in Section III, the three use cases exhibit diverse characteristics (e.g., various file formats, I/O interfaces, metadata) and provenance needs (e.g., file/dataset/attribute lineage, I/O statistics, metadata versioning). We summarize the information tracked by PROV-IO$^+$ in the experiments to meet the provenance needs in Table IV and elaborate them in detail in the following subsections.

Containerized Workflow: Besides experimenting with the classic workflows, we have evaluated the PROV-IO$^+$ framework for one containerized workflow on a supercomputer deployed at Samsung Advanced Institute of Technology (SAIT), which is an HPE Apollo 6500 Gen10 Plus System. For simplicity, we call the system SuperCom in the rest of the paper. Similar to Cori, SuperCom uses Lustre as the parallel file system. Different from Cori, SuperCom's job management is based on a combination of IBM LSF scheduler and Singularity/Apptainer, which is a container runtime designed for HPC environments [24]. A detailed comparison between the two platforms (i.e., Cori and SuperCom) is summarized in Table III.

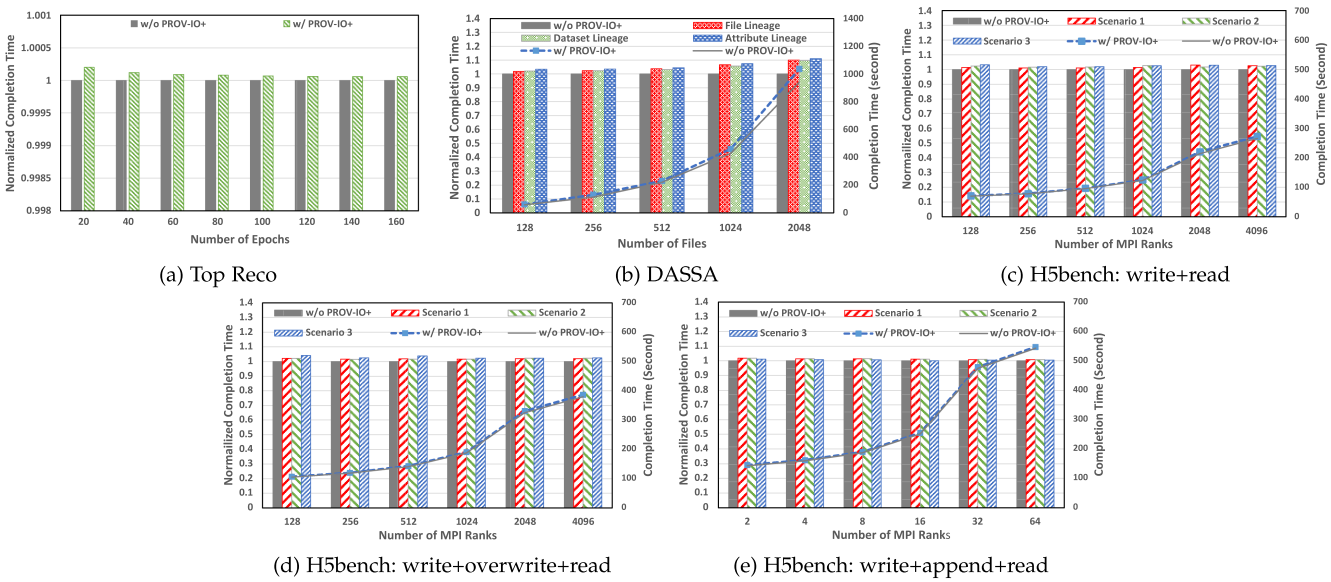
We apply PROV-IO$^+$ to the representative deep learning workflow Megatron-LM [30], which exhibits unique characteristics and provenance needs as discussed in Section III-D and summarized in Table IV. We containerize the workflow through the PROV-IO$^+$ containerizer engine and leverage eight NVIDIA A100 GPUs on SAIT SuperCom to accelerate the training. For clarity, we present the evaluation results on SAIT SuperCom in Section VI-D.

Containerization Impact Analysis: In addition to experimenting on Cori and SuperCom, we have used GCP to study the impact of containerization on provenance tracking. We use GCP because Cori and SuperCom are customized for supporting LBNL's and Samsung's missions respectively, and we cannot modify their runtime environment (e.g., adding or removing HPC container) conveniently. By leveraging GCP, we can build the necessary system environments for running both non-containerized and containerized workflows and conduct a fair comparison on the same infrastructure.

More specifically, we use the GCP Deep Learning virtual machines (VMs) with 32 vCPUs, 120 GB DRAM, and 4 NVIDIA T4 GPUs for the comparison experiments. And we apply PROV-IO$^+$ in two different modes for provenance tracking on two versions of Megatron-LM (i.e., non-containerized and

TABLE IV
PROVENANCE NEEDS AND THE INFORMATION TRACKED BY PROV-IO+ FOR THREE WORKFLOWS

Workflow	Provenance Need	Information Tracked	Komadu?	ProvLake?	PROV-IO+?
Top Reco (Python)	metadata ver. control & mapping	hyperparameter, preselection, training accuracy	No	Yes	Yes
DASSA (C++)	file lineage dataset lineage attribute lineage	program, I/O API, file program, I/O API, dataset program, I/O API, attr	No	No	Yes
H5bench (C)	scenario-1 scenario-2 scenario-3	I/O API I/O API, duration user, thread, program, file	No	No	Yes
Megatron-LM (Python)	ckpt-config consistency	checkpoint info, loss, model configuration	No	Yes	Yes



(a) Top Reco (b) DASSA (c) H5bench: write+read

(d) H5bench: write+overwrite+read (e) H5bench: write+append+read

Fig. 7. Performance of Provenance Tracking. (a) Top Reco. (b) DASSA (with File, Dataset, Attribute Lineages tracked). (c)–(e) H5bench-based workflow under three I/O patterns (i.e., write+read, write+overwrite+read, write+append+read).

containerized) respectively. We discuss the comparison results in Section VI-E.

B. Performance of Provenance Tracking

In case of **Top Reco**, the scientists need the mapping between configurations and the training performance. Therefore, PROV-IO+ tracks three domain-specific items (e.g., model hyperparameters, dataset preselections, and training accuracy) based on the extensible class defined in the PROV-IO+ model. To track the mapping between workflow configuration and training accuracy, we instrument the workflow's training loop with PROV-IO+ APIs and record the training accuracy at the end of each epoch, and add the training accuracy to the provenance graph as a property of configurations. In addition, we vary the number of training epochs to see how the performance scales. Note that Top Reco is a single process workflow.

Fig. 7(a) shows the performance for Top Reco. The y-axis is the normalized completion time (starting with 0.998), while the x-axis is the number of training epoch (roughly equivalent to training time). Since the workflow completion time may vary from a few seconds to a few hours based on the number of training epochs, to make overhead results more comparable across different experiments in the same figure, we use normalized completion time to reflect the performance trend in all tracking overhead discussion throughout the evaluation section. The grey bars are the baseline without provenance, and the green bars show the performance with PROV-IO+ enabled. We can see that the tracking overhead is negligible overall with a maximum of 0.02%. The overhead with a shorter training time is relatively high, which is mostly caused by the latency of Redland. As the number of training epoch increases, the overhead of PROV-IO+ decreases almost linearly because PROV-IO+ tracks a constant amount of information.

In case of **DASSA**, the scientists need the backward lineage of data products in different granularity. As shown in the second column of Table IV, PROV-IO+ tracks the information of user, program, file, dataset, or attribute for different lineage needs based on the PROV-IO+ model (Section IV-A). We follow a similar configuration as the domain scientists' by using 32 compute nodes and up to 2048 input files (1.35 TB in total).

Fig. 7(b) shows the tracking performance for DASSA. The x-axis means the number of input files; the y-axis on the left and right sides show the normalized completion time and the raw completion time (in second), respectively. The grey bars represent the normalized baseline without PROV-IO+, and the red, green and blue bars represent the normalized completion time under three usage scenarios (i.e., "File Lineage", "Dataset Lineage" and "Attribute Lineage") where different provenance granularity are enabled (e.g., for "File Lineage" we enable "program", "I/O API" and "file" tracking). The solid grey line means the average baseline completion time (in second) without

provenance tracking, while the dashed blue line represents the worst case raw completion time with PROV-IO$^+$ under all scenarios.

We can see the max overhead occurred when tracking the attribute lineage of the entire 2048 files, which is about 11%. This is because DASSA heavily relies on HDF5 attributes. To access an attribute, the program first needs to open the file and the dataset containing it, which incurs more I/O operations to track. But overall, PROV-IO$^+$ incurs reasonable overhead in DASSA (from 1.8% to 11%). This is expected because DASSA does not require heavy I/O API tracking. In other words, PROV-IO$^+$ is efficient for tracking the backward lineage in file, dataset, and attribute granularity.

In the **H5bench** based workflow, the scientists need the data usage and I/O statistics in general. We consider three different usage scenarios with different needs. As summarized in Table IV, *scenario-1* tracks the total number of I/O APIs; *scenario-2* tracks both the I/O API count and their duration for bottleneck analysis; *scenario-3* tracks the users and threads that modify the file. Moreover, for each scenario, we consider three different I/O patterns including: *write+read* ("w+r" for short), *write+overwrite+read* ("w+o+r" for short), and *write+append+read* ("w+a+r" for short). In (c) and (d), we run the workflow with 128 to 4096 MPI processes. In (e), since the append operations from a large amount of MPI processes can easily overwhelm the memory buffer for appending and lead to out-of-memory (OOM) errors, we reduce the number of MPI processes (2 to 64). Also, based on the observation that the computation time of many HPC applications may vary from dozens to thousands of seconds per I/O operation, we introduce a relatively modest computation time of 25 seconds per step in the experiments.

Fig. 7(c), (d), (e) show the tracking performance under three different I/O patterns (i.e., "w+r", "w+o+r", "w+a+r"). The x-axis stands for the number of MPI ranks. The left y-axis is the normalized completion time and the right y-axis is the raw completion time in second. The grey bars represent the baseline while the three types of colored bars stand for the performance of different provenance usage scenarios mentioned in Table IV (red for "scenario 1", green for "scenario 2", blue for "scenario 3"). The grey solid line is the average baseline completion time, while the blue dash line is the worst-case raw completion time with PROV-IO$^+$ enabled.

Overall, we find that PROV-IO$^+$ incurs reasonable amount of overhead (i.e., ranging from 0.5% to 4%) even under heavy I/O operations (3.9 TB data with 4096 MPI ranks). In particular, the PROV-IO$^+$ overhead under the "w+a+r" I/O pattern (Fig. 7(c)) is minimal (around 0.5%). This is because the HDF5 I/O operation under this pattern takes more computation time than under the other two patterns to determine the append offset and memory range, which makes the PROV-IO$^+$ overhead more negligible. Also, by comparing scenario-1 and scenario-2, we find that tracking the I/O API duration introduce little additional overhead. This is reasonable because the timing information can be piggybacked with the I/O API tracking which dominates the overall tracking time.

C. Storage Requirements

The storage requirement of PROV-IO$^+$ is directly related to the amount and the class of information tracked. Specifically, the storage overhead may increase in two ways: (1) the size of a single provenance record may increase (e.g., adding timing information will increase size of an I/O API record); (2) the total number of records in a provenance file may increase (e.g., tracking thread information will create a number of thread records). We summarize the storage performance of PROV-IO$^+$ for the three workflows in Fig. 8.

Fig. 8(a) shows the Top Reco case. The x-axis represents the number of epochs and the y-axis is the provenance size (KB). We can see that the provenance size is negligible. This is because PROV-IO$^+$ allows users to specify the target provenance precisely without incurring unnecessary overhead. It also scales linearly since the number of new nodes added to provenance graph is the same as the increment in training epochs.

Fig. 8(b) shows the DASSA case. The x-axis is the number of input files while the y-axis represents the provenance size (MB). Lines in three different colors represent File Lineage, Dataset Lineage and Attribute Lineage, respectively. We can see that the storage requirement varies from 40 MBs (with 128 input files) to about 800 MBs (with 2048 files) with linear scalability (note that the x-axis increases by a multiple of 2). Although DASSA heavily relies on attributes, the storage overhead in the three usage scenarios is similar. This is because I/O API is still the dominant part in all scenarios. Even though the number of file and dataset is far less than attribute in DASSA input data, when compared to number of APIs involved in the workflow, their contribution to storage overhead is insignificant.

Fig. 8(c), (d), (e) shows the H5bench-based workflow with three different I/O patterns. The x-axis represents number of MPI ranks and the y-axis stands for provenance size in MBs. Note that x-axis also increases by a multiple of 2. Lines in three different colors represents three different provenance usage scenarios (Table IV). We can see the provenance size varies from a few KBs to 168 MBs. Among the three I/O patterns, "w+o+r" has the highest storage overhead under usage scenario 2. This is because the pattern includes one more I/O application (i.e., overwrite) than "w+r" and has much more MPI processes contributing to provenance graph than "w+a+r". Moreover, scenario 2 also has the largest amount of tracked information (I/O API and their duration). Note that the storage overhead in this workflow also scales linearly.

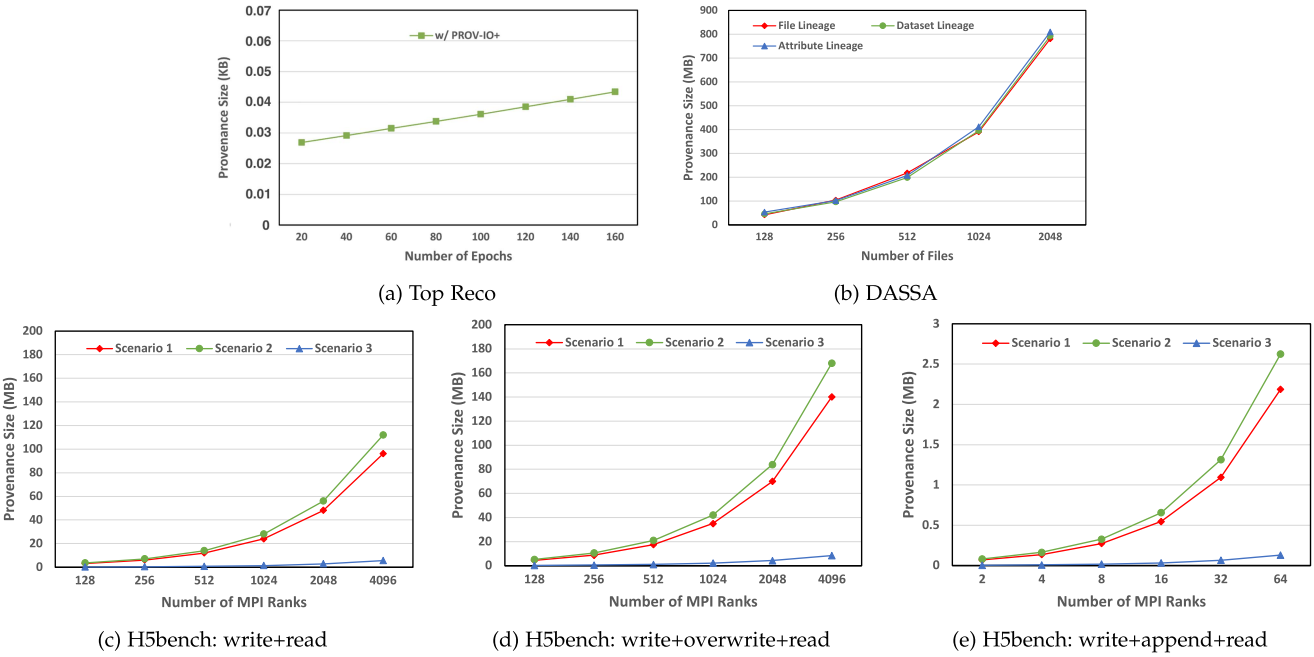
In summary, because of the flexibility of the fine-grained PROV-IO$^+$ model, PROV-IO$^+$'s storage overhead is reasonable for all the use cases evaluated.

D. Experiments With Containerized Workflow

In this section, we introduce our experiments with a containerized workflow (i.e., Megatron-LM [30]) on the S-SuperCom.

Megatron-LM supports multiple pretraining models (Section III-D), and we configure Megatron-LM to use GPT-2 as the pretraining model in this set of experiments based on the need of the domain scientists. The training dataset is WikiText103 [64] which is provided by the Megatron-LM authors [30]. We enable both model parallelism and data parallelism (Section III-D) in the workflow for experiments.

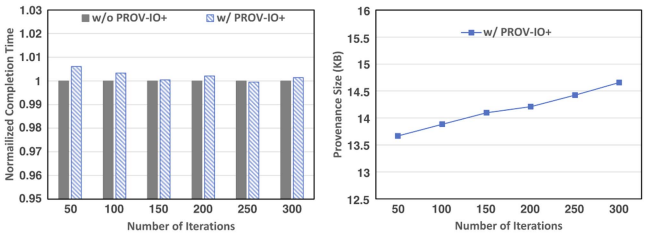
As mentioned in Section III-D, the major provenance need in the Megatron-LM use case is to ensure the consistency between the checkpoint and the workflow configurations. Therefore, we track detailed checkpoint information pertaining to a pretraining process (e.g., the path of the checkpoint file) as well as a variety of relevant configuration parameters (e.g., the number of GPUs, the batch size). The configuration information is



(a) Top Reco (b) DASSA

(c) H5bench: write+read (d) H5bench: write+overwrite+read (e) H5bench: write+append+read

Fig. 8. Storage of provenance tracking. (a) Top Reco. (b) DASSA (with File, Dataset, Attribute Lineages tracked). (c)–(e) H5bench-based workflow under three I/O patterns (i.e., write+read, write+overwrite+read, write+append+read).



(a) Tracking Performance. (b) Storage Requirement.

Fig. 9. Performance of PROV-IO+ on Megatron-LM with checkpoint path, training loss and configuration tracked.

tracked once at the beginning of workflow execution as the information remains invariant throughout the workflow execution, while the checkpoint information is tracked transparently by instrumenting PyTorch at the end of the workflow execution. In addition, we record the GPT-2 training loss at the end of each training iteration. We change the number of training iterations in the experiments to measure how the tracking performance and storage requirement scales. We report the measurement results as follows.

Fig. 9(a) shows the provenance tracking performance. The y-axis is the normalized workflow completion time, and the x-axis is the number of training iteration. We use the grey bar to represent the baseline without provenance tracking, and the blue bar stands for workflow completion time with provenance enabled. For each experiment, we repeat it five times and calculate the average (mean) performance value to eliminate slight outliers caused by cross-zone network fluctuations. The result shows that the maximum tracking overhead is about 0.6% when the training iteration is set to 50. When the number of iteration increases, PROV-IO+'s tracking overhead tends to be negligible, which implies that PROV-IO+ is scalable in terms of tracking performance.

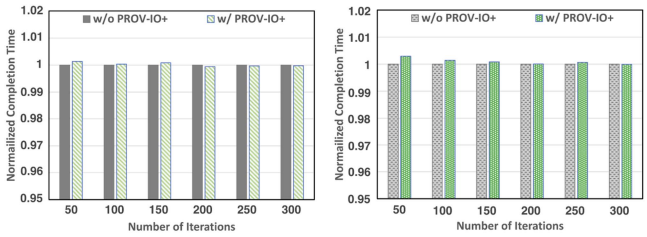
Fig. 9(b) shows the storage requirement of tracking Megatron-LM with PROV-IO+. The y-axis is the provenance size (KB), and the x-axis is the number of training iteration. The result shows that, to track the checkpoint-configuration consistency information, the provenance size is negligible in general (e.g., less than 15 KB in all experiments). The size of the provenance information scales almost linearly as the number of training iterations increases, mainly because the training loss is recorded at the end of each training iteration.

In summary, to track the necessary provenance for maintaining the checkpoint-configuration consistency in containerized Megatron-LM, PROV-IO+ introduces small tracking and negligible storage consumption.

E. Impact of Containerziation on Provenance Tracking

In this section, we analyze the impact of containerization on PROV-IO+'s provenance collection. As mentioned in Section VI-A, we leverage the GCP platform for the comparison experiments because we can setup different runtime environments for both containerized and non-containarized workflows on GCP. We apply PROV-IO+ in two different modes for tracking two versions of Megatron-LM (i.e., non-containerized and containerized) on GCP respectively.

To validate the impact of containerization, we execute the non-containerized version of Megatron-LM workflow and the containerized version separately on different GCP VMs. This is to ensure that there is no interference between the executions of the two versions. The provenance information tracked is the same as described in Section VI-D. We reduce the scale of the workflow to meet the VM's resource constraints (e.g., vCPUs and memory).



(a) Non-containerized workflow (b) Containerized workflow

Fig. 10. Comparison of PROV-IO$^+$ tracking overhead in non-containerized and containerized Megatron-LM.

TABLE V
BASIC CHARACTERISTICS OF THREE FRAMEWORKS

	Komadu	**ProvLake**	**PROV-IO$^+$**
Base model	PROV-DM	PROV-DM	PROV-DM
Language	Java	Python	C/C++,Python,Java
Transparency	No	No	Hybrid

The performance of PROV-IO$^+$ on non-containerized Megatron-LM and containerized Megatron-LM are shown in Fig. 10(a) and (b), respectively. In both cases, the y-axis is the normalized workflow completion time, and the x-axis is the number of training iteration. By comparing Fig. 10(a) and (b), we can see that in both cases PROV-IO$^+$ incurs little overhead, especially when the number of iterations is large. This suggests that containerization has little impact on PROV-IO$^+$, and both modes of PROV-IO$^+$ can support provenance tracking efficiently.

In conclusion, our experiments on three different platforms (i.e., Cori in Section VI-B, SuperCom in Section VI-D, and GCP Section VI-E) shows that PROV-IO$^+$'s provenance tracking performance has little dependence on the platforms, and the overhead is consistently low across different execution platforms.

F. Comparison With Other Frameworks

In this section, we compare PROV-IO$^+$ to state-of-the-art provenance systems. Table V shows the basic characteristics of Komadu [65], ProvLake [20], and PROV-IO$^+$. We can see that all three frameworks are derived from the base PROV-DM model, which makes the comparison fair. On the other hand, Komadu only supports Java programs and ProvLake only supports Python, which makes them incompatible with many C/C++ based scientific workflows (e.g., DASSA and H5bench). Note that PROV-IO$^+$'s C/C++ interface is designed for integration with major HPC I/O libraries. Once the I/O library is integrated with PROV-IO$^+$ (e.g., HDF5), the provenance support is mostly transparent to the workflow users, i.e., users can control the rich provenance features through a configuration file without manually modifying their source code with APIs. Neither Komadu nor ProvLake support such capability or transparency.

Since ProvLake has outperformed Komadu based on a previous study [17], we focus on the comparison with ProvLake. Because ProvLake does not support C/C++ workflows, we cannot apply it to DASSA and H5bench. Therefore, we compare the two provenance tools using Python-based Top Reco in the rest of this section.

Different from PROV-IO$^+$ which is I/O-centric, ProvLake is 'process-oriented'. Specifically, ProvLake creates records based on the execution steps of a workflow, and the provenance data are maintained as attribute or property of individual steps. On the contrary, PROV-IO$^+$ is not limited to the execution steps of the workflow. For example, it can track a task in the workflow, an I/O operation invoked by a task, a data object involved in the I/O operation, etc., all of which are further correlated via the relations defined by the PROV-IO$^+$ model (Section IV-A). Such flexibility and richness is not available in ProvLake.

To make the comparison with ProvLake fair, we use the same instrument points in the Top Reco workflow for both tools. Specifically, we instrument Top Reco at its GNN training loop and track the training accuracy at the end of each epoch to corresponding provenance records. Since the workflow configuration is never changed during the entire workflow, we only add it to ProvLake's record once at the beginning of the workflow. In addition, to be representative, we track three different numbers of configurations (i.e., 20, 40, and 80).

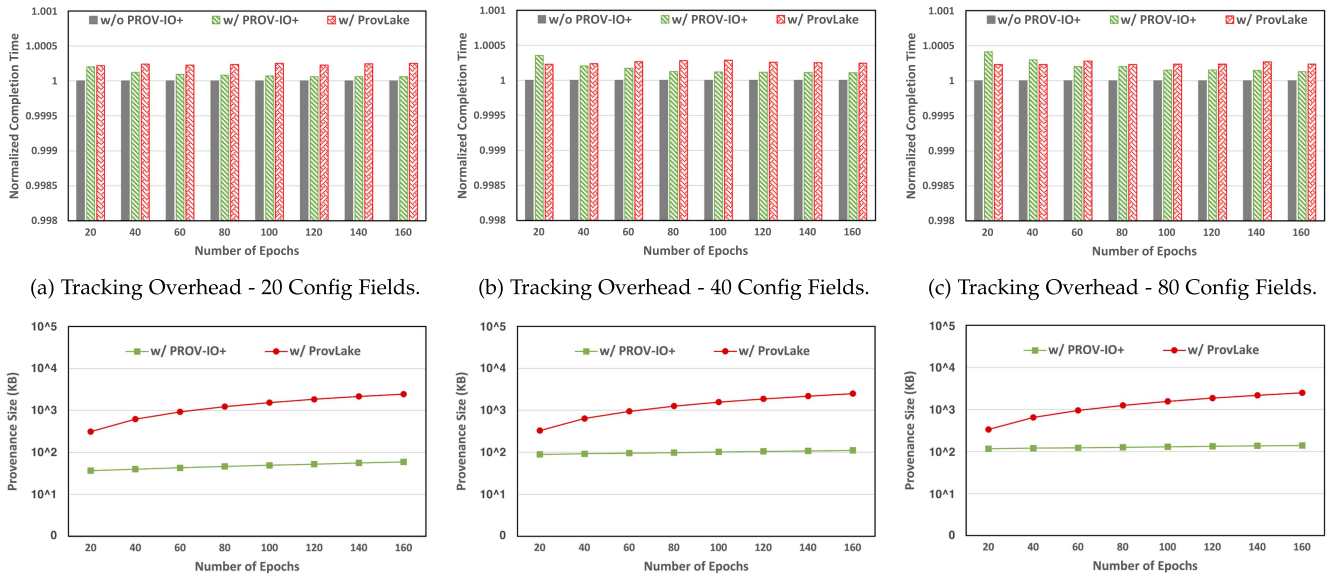
Fig. 11(a), (b), (c) compares the provenance tracking performance of the two systems where y-axis is normalized completion time. Fig. 11(d), (e), (f) shows the storage overhead where y-axis is size in KB. In all figures x-axis is the number of configurations. In Fig. 11(a), (b), (c), grey bars stand for the baseline without provenance tracking, green bars show the normalized performance with PROV-IO$^+$, and red bars show the performance with ProvLake. In Fig. 11(d), (e), (f), green lines stand for PROV-IO$^+$ provenance file size and red lines stand for ProvLake provenance file size.

As shown in Fig. 11(a), (b), (c), both frameworks incur negligible tracking overhead (e.g., less than 0.025%) and the PROV-IO$^+$ overhead is even lower than ProvLake for most cases. Similarly, as shown in Fig. 11(d), (e), (f), PROV-IO$^+$ always incurs less storage overhead, regardless of the number of configuration fields tracked. The disparity in provenance models between PROV-IO$^+$ and ProvLake is primarily responsible for the result. PROV-IO$^+$ forms a provenance graph for data concepts in the workflow, whereas ProvLake concentrates on the workflow. In this scenario, ProvLake gathers additional information for each training cycle, which is unnecessary for workflow metadata version control.

G. Query Effectiveness

As mentioned in Section V, PROV-IO$^+$ supports provenance query with visualization. Table VI summarizes the queries used to answer the diverse provenance needs of the three workflow cases. We can see that the provenance can be queried effectively and efficiently using a few simple SPARQL statements in general. Since the number of queries involved is small, the query time overhead is negligible in our experiments. We discuss each case in more details below.

In DASSA, to get the backward lineage of a data product, we can start with the program which generated the data product and look for its input data. The same procedure can be repeated as needed. For example, DASSA may convert "WestSac.tdms" into "WestSac.h5" with program "tdms2h5", and then use "decimate" to process "WestSac.h5" into data product "decimate.h5". To get the backward lineage of "decimate.h5", in the query, we first retrieve with keywords "decimate.h5 prov:wasAttributedTo ?program" to locate program "decimate". Next, in the same query, we add statement "?file wasAttributedTo ?program" to retrieve that program's input file "WestSac.h5" which is the first level predecessor of "decimate.h5". We can further expand the



(a) Tracking Overhead - 20 Config Fields. (b) Tracking Overhead - 40 Config Fields. (c) Tracking Overhead - 80 Config Fields.

(d) Storage Overhead - 20 Config Fields. (e) Storage Overhead - 40 Config Fields. (f) Storage Overhead - 80 Config Fields.

Fig. 11. Performance comparison between PROV-IO+ and ProvLake.

TABLE VI
EXAMPLE QUERIES

Workflow	Provenance Need	Query Statement (SPARQL)	# of Statements in Query
DASSA	file/dataset/attribute lineage	1: data_object_a prov:wasAttributedTo ?program. 2: ?data_object prov:wasAttributedTo ?program; 3: provio:wasReadBy ?IO_API.	3*N (where N is backward propagation steps)
H5bench	scenario-1	4: ?IO_API a provio:ioAPI.	1
	scenario-2	5: ?IO_API a provio:ioAPI; 6: provio:elapsed ?duration.	2
	scenario-3	7: file_a prov:wasAttributedTo ?program. 8: ?program prov:actedOnBehalfOf ?thread. 9: ?thread prov:actedOnBehalfOf ?user.	3
Top Reco	metadata version control & mapping	10: ?configuration ns1:Version ?version; 11: provio:hasAccuracy ?accuracy.	2
Megatron-LM	ckpt-config consistency	12: ?batch_size ns1:hasValue 256; 13: prov:influenced ?checkpoint_path	2

The diverse provenance needs can be satisfied by a few simple queries effectively.

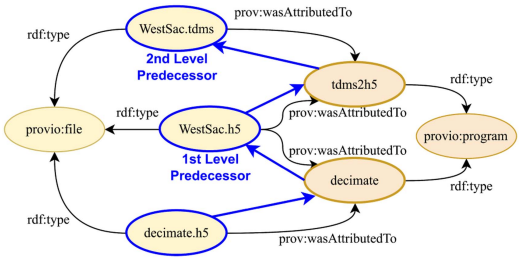


Fig. 12. Example of DASSA data lineage by PROV-IO+. The graph follows the PROV-IO+ model; the data lineage is highlighted in blue.

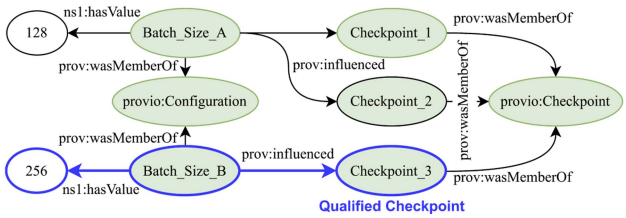


Fig. 13. Example of Megatron-LM query by PROV-IO+. The graph follows the PROV-IO+ model; the query path is highlighted in blue.

query by adding similar statements to locate "decimate.h5"'s earlier predecessors (e.g., "WestSac.tdms").

As summarized in Table VI, for each backward step, we only need three query statements. Fig. 12 shows the visualization of this example, which follows the PROV-IO+ provenance model (Section IV-A) and highlights the queried data lineage in blue. Other types of lineages (e.g., dataset and attribute) can be queried and visualized in the same way.

Similarly, in H5bench, we have three types of provenance needs (i.e., the scenarios described in Section VI-B) which can be answered using 1, 2, 3 SPARQL statements respectively. In Top Reco, the metadata versioning and mapping information can be queried in 2 statements. Note that the provenance needs are diverse across the real use cases, but the number of queries needed is consistently small. This elegant result suggests that PROVI-IO+ is effective for scientific data on HPC systems.

In case of Megatron-LM, users want to identify the checkpoint which is consistent with the configuration for the follow-up

training process. Fig. 13 shows a scenario where there are two different batch sizes used during the previous pretraining (i.e., "Batch_Size_A" is 128, and "Batch_Size_B" is 256), and there are three checkpoints generated based on the two batch sizes (i.e., "Checkpoint_1", "Checkpoint_2", "Checkpoint_3"). Assume the user wants to continue a GPT pretraining process which has a batch size of 256 with one of the existing checkpoints (i.e., "Checkpoint_3"), s/he can query the provenance with as few as 2 lines of SPARQL statements, as shown in the last row of Table VI. Moreover, the user can also add advanced conditions to the query to filter out the feasible checkpoint with the best quality (e.g., a checkpoint with certain training loss).

VII. Discussion

The design of the PROV-IO$^+$ tool is driven by the needs of the domain scientists using four scientific workflows. Given the diversity of science, it is likely that the prototype cannot directly address the unique provenance queries of all scientists. We plan to collaborate with more domain scientists to identify additional needs and refine PROV-IO$^+$ accordingly. For example, researchers from INRIA [66] are interested in porting PROV-IO$^+$ on their edge devices with limited hardware resource. Similarly, HPE [67] researchers are interested in integrating PROV-IO$^+$ to their provenance solutions. We are in communication with the researchers to extend the real-world impact of PROV-IO$^+$ further.

Similarly, while the current prototype supports POSIX and HDF5 I/O transparently and is extensible by design, there are other popular I/O systems in HPC (e.g., ADIOS [32]) which we have not integrated yet. We leave the integration with other I/O libraries as future work.

In addition, there are other important aspects of provenance (e.g., security [68]) which cannot be ignored in practice. We hope that our efforts and the resulting open-source tool can facilitate follow-up research in the communities and help address the grand challenge of provenance support for scientific data in general.

VIII. Related Work

Database Provenance: Historically, provenance has been well studied in databases to understand the causal relationship between materialized views and table updates [6], [69]. The concept has also been extended to other usages [8], [70]. In general, database provenance may leverage the well-defined relational model and the relatively strict transformations to capture precise provenance within the system [71], which is not applicable for general software. On the other hand, some query optimizations (e.g., provenance reduction [72]) could potentially be applied to PROV-IO$^+$. Therefore, PROV-IO$^+$ and these tools are complementary.

OS-Level Provenance: Great efforts have also been made to capture provenance at the operating system (OS) level [11], [12], [50]. For example, PASS [11], [12] intercepts system calls via custom kernel modules for inferring data dependencies. Similarly to these efforts, PROV-IO$^+$ recognizes the importance of I/O syscalls. But different from PASS, PROV-IO$^+$ is non-intrusive to the OS kernel. Moreover, PROV-IO$^+$ leverages the unique characteristics of HPC workflows and systems to meet the needs of domain scientists, while PASS is largely inapplicable in this context. More specifically, we elaborate on five key differences as follows:

(1) *Provenance Model:* PROV-IO$^+$ follows the W3C specifications to represent rich provenance information in a relational model (Section IV-A). In contrast, PASS follows the conventional logging mechanism without a general relational model, which limits its capability of capturing and describing complex provenance. For example, PASS has to establish the dependencies among events via a kernel-level logger (i.e., 'Observer' [12]) which cannot interpret the semantics or relations of HPC I/O library events. Consequently, PASS can only answer relatively limited queries (e.g., ancestor of a node [12]) instead of the rich lineage defined in W3C.

(2) *System Architecture:* PROV-IO$^+$ is a user-level solution designed for the HPC environment (Section IV-B). In contrast, PASS heavily relies on customized kernel modules to achieve its core functionalities. This kernel-based architecture makes PASS incompatible with modern HPC systems. For example, neither the PASTA file system (in PASS [11]) nor the Lasagna file system (in PASSv2 [12]) is compatible with the Lustre PFS dominant in HPC. In other words, translating the core functionalities of PASS to HPC systems would require substantial efforts (if possible at all), and the implications on performance and scalability is unclear.

(3) *Granularity:* PROV-IO$^+$ can handle fine-grained I/O provenance which is critical for understanding HPC workflows (e.g., the lineage of an attribute of an HDF5 file), while PASS collects relatively coarse-grained events (e.g., access to an entire file). Container-based approaches for fine-grained provenance collection have also been proposed [73], [74], which we consider as a orthogonal approach to PROV-IO$^+$.

(4) *Tracking APIs:* By embedding in popular HPC I/O libraries, PROV-IO$^+$ does not require modifying the source code to track I/O provenance. In contrast, to use PASS, users must consider how to apply six low-level calls (e.g., `pass_read`, `pass_mkobj` [12]) to the target applications.

(5) *Storage & Query:* Based on the well-defined model, PROV-IO$^+$ stores provenance as RDF triples backed by the parallel file system. In contrast, PASS relies on its own local file system to generate provenance as local logs. The storage representation directly affects the user query capability. For example, PROV-IO$^+$ supports querying RDF triples via SPARQL [61], while PASS only supports a special Path Query Language which is much less popular today.

In summary, while PROV-IO$^+$ is partially inspired by the seminal PASS designed more than a decade ago, the two works are different due to the different goals and contexts. Therefore, we view PASS and PROV-IO$^+$ as complementary.

Workflow & Application Provenance: Provenance models or systems for workflows and/or applications have also been explored [18], [20], [21], [75]. For example, Karma [21] describes a model with a hierarchy of 'workflow-service-application-data'. However, the model is designed for the cloud environment and cannot cover diverse HPC needs (e.g., HDF5 attributes, MPI ranks). PROV-ML [18] is a series of well-defined specifications for machine learning workflows. Different from PROV-ML, PROV-IO$^+$ is designed for general HPC workflows. IBM ProvLake [20] is a lineage data management system capable of capturing data provenance across programs. Unlike PROV-IO$^+$, ProvLake always require users to modify the source code using its special APIs, which severely limits its usage and scalability for complicated HPC workflows. Similar to PROV-IO$^+$, there are a few provenance capturing tools using DBMS to store queriable provenance data, but they do not follow any widely used provenance models [76], [77], [78].

Other Usage of Provenance: Provenance has been applied to other venues. For example, MOLLY uses lineage-driven fault injection to expose bugs in fault-tolerant protocols [79]. There have been a multitude of domain-specific or application-specific provenance and ontology management implementations. However, they do not capture the I/O access information that PROV-IO+ manages. We believe the comprehensive provenance information enabled by PROV-IO+ can also be leveraged to stimulate several data quality and storage optimizations, which we leave as future work.

Non-Provenance I/O Tools & Analysis: In addition, great efforts have been made to manage workflows [80], [81] and/or leverage I/O events for reliability [38], [39], [82], [83], [84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103]. While they are effective for their original goals, they are insufficient to address provenance needs in general due to a number of reasons: (1) no relational model to support managing rich provenance (e.g., various relations in [31]); (2) agnostic to the fine-grained semantics in HPC I/O libraries (e.g., HDF5 attributes); (3) little portability across different I/O libraries or environments; (4) no programmable interface to specify customized provenance needs.

IX. Conclusion & Future Work

We have introduced a provenance tool called PROV-IO+ for scientific data on HPC systems. Experiments with representative HPC workflows show that PROV-IO+ can address diverse provenance needs with reasonable overhead. In the future, we will address the limitations mentioned in Section VII. Moreover, in the Top Reco case studied in this paper, the domain scientists would like to identify the best configurations across multiple runs of the workflow. In other words, there is a need of provenance across multiple executions of the same workflow. Similar cross-workflow provenance may be needed when multiple different workflows cooperate to process shared datasets, which requires additional modeling and interface to bridge the semantic gap between workflows. We would like to investigate such complex multi-workflow scenario as well. In addition, we observe that diagnosing the correctness and performance anomalies in HPC systems is increasingly challenging due to the complexity (e.g., a single SSD failure may cause the "blast radius" problem due to system dependencies [88], [100]), and we will apply PROV-IO+ to address such open challenges. Overall, we believe that PROV-IO+ represents a promising direction toward ensuring the rigorousness and trustworthiness of scientific data management.

Acknowledgment

We thank the anonymous reviewers and Matthew Wolf, Xiangyang Ju, James Loo, Jim Wayda for their invaluable feedback. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors. This manuscript has been authored by an author at Lawrence Berkeley National Laboratory under Grant DE-AC02-05CH11231 with the U.S. Department of Energy. The U.S. Government retains, and the publisher, by accepting the article for publication, acknowledges, that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

References

[1] K. M. Tolle, D. S. W. Tansley, and A. J. G. Hey, "The fourth paradigm: Data-intensive scientific discovery," *Proc. IEEE*, vol. 99, no. 8, pp. 1334–1337, Aug. 2011.
[2] M. D. Wilkinson et al., "The FAIR guiding principles for scientific data management and stewardship," *Sci. Data*, vol. 3, pp. 1–9, 2016.
[3] Fair principles. Accessed: Feb. 20, 2024. [Online]. Available: https://www.go-fair.org/fair-principles/
[4] M. Milton and A. Possolo, "Trustworthy data underpin reproducible research," *Nat. Phys.*, vol. 16, pp. 117–119, 2020.
[5] B. Dong, V. Tribaldos, X. Xing, S. Byna, J. Ajo-Franklin, and K. Wu, "DASSA: Parallel DAS data storage and analysis for subsurface event detection," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2020, pp. 254–263.
[6] P. Buneman, S. Khanna, and W. C. Tan, "Why and where: A characterization of data provenance," in *Proc. 8th Int. Conf. Database Theory*, 2001, pp. 316–330.
[7] X. Niu, B. Glavic, D. Gawlick, Z. H. Liu, V. Krishnaswamy, and V. Radhakrishnan, "Interoperability for provenance-aware databases using PROV and JSON," in *Proc. 7th USENIX Workshop Theory Pract. Provenance*, 2015, pp. 6–6.
[8] P. Senellart, "Provenance and probabilities in relational databases," *ACM SIGMOD Rec.*, vol. 46, pp. 5–15, 2018.
[9] Z. Miao, Q. Zeng, B. Glavic, and S. Roy, "Going beyond provenance: Explaining query answers with pattern-based counterbalances," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 485–502.
[10] Z. Wang et al., "A provenance storage method based on parallel database," in *Proc. 2nd Int. Conf. Inf. Sci. Control Eng.*, 2015, pp. 63–66.
[11] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proc. USENIX Annu. Tech. Conf.*, 2006, pp. 43–56.
[12] K.-K. Muniswamy-Reddy et al., "Layering in provenance systems," in *Proc. USENIX Annu. Tech. Conf.*, 2009, pp. 10–10.
[13] E. Jandre, B. Diirr, and V. Braganholo, "Provenance in collaborative in silico scientific research: A survey," *ACM SIGMOD Rec.*, vol. 49, pp. 36–51, 2020.
[14] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *ACM SIGMOD Rec.*, vol. 34, pp. 31–36, 2005.
[15] S. B. Davidson and J. Freire, "Provenance and scientific workflows: Challenges and opportunities," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 1345–1350.
[16] I. Suriarachchi, S. Withana, and B. Plale, "Big provenance stream processing for data intensive computations," in *Proc. IEEE 14th Int. Conf. eScience*, 2018, pp. 245–255.
[17] R. Souza et al., "Efficient runtime capture of multiworkflow data using provenance," in *Proc. 15th Int. Conf. eScience*, 2019, pp. 359–368.
[18] R. Souza et al., "Provenance data in the machine learning lifecycle in computational science and engineering," in *Proc. IEEE/ACM Workflows Support Large-Scale Sci.*, 2019, pp. 1–10.
[19] The PROV data model - W3C. Accessed: Feb. 20, 2024. [Online]. Available: https://www.w3.org/TR/prov-overview/
[20] R. Souza et al., "Workflow provenance in the lifecycle of scientific machine learning," *Concurrency Comput. Pract. Experience*, vol. 34, 2022, Art. no. e6544.
[21] Y. L. Simmhan, B. Plale, and D. Gannon, "A framework for collecting provenance in data-centric scientific workflows," in *Proc. IEEE Int. Conf. Web Serv.*, 2006, pp. 427–436.
[22] B. Howe et al., "End-to-end eScience: Integrating workflow, query, visualization, and provenance at an ocean observatory," in *Proc. IEEE 4th Int. Conf. eScience*, 2008, pp. 127–134.
[23] Resource description framework. Accessed: Feb. 20, 2024. [Online]. Available: https://www.w3.org/RDF/
[24] Apptainer/singularity. Accessed: Feb. 20, 2024. [Online]. Available: https://apptainer.org
[25] Cori at LBNL. Accessed: Feb. 20, 2024. [Online]. Available: https://www.nersc.gov/systems/cori/
[26] Samsung advanced institute of technology. Accessed: Feb. 20, 2024. [Online]. Available: https://www.sait.samsung.co.kr/saithome/main/main.do
[27] Google cloud platform. Accessed: Feb. 20, 2024. [Online]. Available: https://cloud.google.com
[28] X. Allison, W. Haichen, and J. Xiangyang, A graph neural network-based top quark reconstruction package. Accessed: Feb. 20, 2024. [Online]. Available: https://indico.cern.ch/event/932415/contributions/3918265/attachments/2086561/3505362/GNN_Top_Reco_-_Allison_Xu.pdf

[29] H5bench. Accessed: Feb. 20, 2024. [Online]. Available: https://www.hdfgroup.org/solutions/hdf5/

[30] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019. [Online]. Available: https://arxiv.org/abs/1909.08053

[31] P. Missier, K. Belhajjame, and J. Cheney, "The W3C PROV family of specifications for modelling provenance metadata," in *Proc. 16th Int. Conf. Extending Database Technol.*, 2013, pp. 773–776.

[32] Adios. Accessed: Feb. 20, 2024. [Online]. Available: https://www.olcf.ornl.gov/center-projects/adios/

[33] HDF5. Accessed: Feb. 20, 2024. [Online]. Available: https://https://www.hdfgroup.org/solutions/hdf5/

[34] NetCDF. Accessed: Feb. 20, 2024. [Online]. Available: https://www.unidata.ucar.edu/software/netcdf/

[35] Automatic library tracking database at NERSC. Accessed: Feb. 20, 2024. [Online]. Available: https://www.nersc.gov/assets/altdatNERSC.pdf

[36] T. Li, Q. Koziol, H. Tang, J. Liu, and S. Byna, "H5Prov: I/O performance analysis of science applications using HDF5 file-level provenance," *Cray User Group (CUG'19) Conf.*, 2019.

[37] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the HDF5 technology suite and its applications," in *Proc. EDBT/ICDT 2011 Workshop Array Databases*, 2011, pp. 36–47.

[38] Darshan, HPC I/O characterization tool. Accessed: Feb. 20, 2024. [Online]. Available: https://www.mcs.anl.gov/research/projects/darshan/

[39] S. Yellapragada, C. Wang, and M. Snir, "Verifying IO synchronization from MPI traces," in *Proc. IEEE/ACM 6th Int. Parallel Data Syst. Workshop*, 2021, pp. 41–46.

[40] A. Vaswani et al., "Attention is all you need," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.

[41] D. Narayanan et al., "Efficient large-scale language model training on GPU clusters using Megatron-LM," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2021, pp. 1–15.

[42] A. Radford et al., "Improving language understanding by generative pre-training," 2018.

[43] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

[44] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," 2019. [Online]. Available: https://arxiv.org/abs/1910.10683

[45] Docker. Accessed: Feb. 20, 2024. [Online]. Available: https://www.docker.com

[46] Kubernetes. Accessed: Feb. 20, 2024. [Online]. Available: https://kubernetes.io

[47] PyTorch. Accessed: Feb. 20, 2024. [Online]. Available: https://pytorch.org

[48] NCCL. Accessed: Feb. 20, 2024. [Online]. Available: https://developer.nvidia.com/nccl

[49] Ext4. Accessed: Feb. 20, 2024. [Online]. Available: https://ext4.wiki.kernel.org/index.php/Main_Page

[50] A. Gehani and D. Tariq, "SPADE: Support for provenance auditing in distributed environments," in *Proc. 13th Int. Middleware Conf.*, 2012, pp. 101–120.

[51] R. Han, S. Byna, H. Tang, B. Dong, and M. Zheng, "PROV-IO: An I/O-centric provenance framework for scientific data on HPC systems," in *Proc. 31st Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2022, pp. 213–226.

[52] Gotcha v1.0.2. Accessed: Feb. 20, 2024. [Online]. Available: https://github.com/LLNL/GOTCHA

[53] Redland RDF. Accessed: Feb. 20, 2024. [Online]. Available: https://librdf.org

[54] Apache Jena. Accessed: Feb. 20, 2024. [Online]. Available: https://jena.apache.org

[55] Neo4j. Accessed: Feb. 20, 2024. [Online]. Available: https://neo4j.com

[56] Blazegraph. Accessed: Feb. 20, 2024. [Online]. Available: https://blazegraph.com

[57] Apache Rya. Accessed: Feb. 20, 2024. [Online]. Available: https://rya.apache.org

[58] Anzographdb. Accessed: Feb. 20, 2024. [Online]. Available: https://cambridgesemantics.com/anzograph/

[59] Terse RDF triple language. Accessed: Feb. 20, 2024. [Online]. Available: https://www.w3.org/TR/turtle/

[60] N-triples. Accessed: Feb. 20, 2024. [Online]. Available: https://www.w3.org/TR/n-triples/

[61] SPARQL query language for RDF. Accessed: Feb. 20, 2024. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/

[62] Graphviz. Accessed: Feb. 20, 2024. [Online]. Available: https://graphviz.org

[63] IBM spectrum LSF session scheduler. Accessed: Feb. 20, 2024. [Online]. Available: https://www.ibm.com/docs/en/spectrum-lsf/10.1.0?topic=lsf-session-scheduler

[64] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016. [Online]. Available: http://arxiv.org/abs/1609.07843

[65] I. Suriarachchi, Q. Zhou, and B. Plale, "Komadu: A capture and visualization system for scientific data provenance," *J. Open Res. Softw.*, vol. 3, pp. e4–e4, 2015.

[66] Inria kerdata team. Accessed: Feb. 20, 2024. [Online]. Available: https://team.inria.fr/kerdata/

[67] HPE AI research lab. Accessed: Feb. 20, 2024. [Online]. Available: https://www.hpe.com/us/en/hewlett-packard-labs.html

[68] A. Bates, D. Tian, K. R. B. Butler, and T. Moyer, "Trustworthy whole-system provenance for the Linux kernel," in *Proc. 24th USENIX Conf. Secur. Symp.*, 2015, pp. 319–334.

[69] Y. Cui, J. Widom, and J. L. Wiener, "Tracing the lineage of view data in a warehousing environment," *ACM Trans. Database Syst.*, vol. 25, pp. 179–227, 2000.

[70] J. Widom, "Trio: A system for integrated management of data, accuracy, and lineage," in *Proc. 2nd Biennial Conf. Innov. Data Syst. Res.*, 2005, pp. 262–276.

[71] L. Carata et al., "A primer on provenance," *Commun. ACM*, vol. 57, pp. 52–60, 2014.

[72] D. Deutch, Y. Moskovitch, and N. Rinetzky, "Hypothetical reasoning via provenance abstraction," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 537–554.

[73] P. Olaya et al., "Building trust in earth science findings through data traceability and results explainability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 2, pp. 704–717, Feb. 2023.

[74] P. Olaya, J. Lofstead, and M. Taufer, "Building containerized environments for reproducibility and traceability of scientific workflows," 2020, *arXiv:2009.08495*.

[75] Q. Zhou, D. Ghoshal, and B. Plale, "Study in usefulness of middleware-only provenance," in *Proc. IEEE 10th Int. Conf. eScience*, 2014, pp. 215–222.

[76] Braid-DB. Accessed: Feb. 20, 2024. [Online]. Available: https://github.com/ANL-Braid/DB/

[77] Chimbuko. Accessed: Feb. 20, 2024. [Online]. Available: https://github.com/CODARcode/Chimbuko

[78] J. Logan et al., "A vision for managing extreme-scale data hoards," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1806–1817.

[79] P. Alvaro, J. Rosen, and J. M. Hellerstein, "Lineage-driven fault injection," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 331–346.

[80] Apache taverna. Accessed: Feb. 20, 2024. [Online]. Available: https://incubator.apache.org/projects/taverna.html

[81] Effis. Accessed: Feb. 20, 2024. [Online]. Available: https://github.com/wdmapp/effis

[82] M. Zheng et al., "Torturing databases for fun and profit," in *Proc. 11th USENIX Symp. Operating Syst. Des. Implementation*, 2014, pp. 449–464.

[83] J. Cao et al., "PFault: A general framework for analyzing the reliability of high-performance parallel file systems," in *Proc. Int. Conf. Supercomput.*, 2018, pp. 1–11.

[84] R. Han et al., "A study of failure recovery and logging of high-performance parallel file systems," *ACM Trans. Storage*, vol. 18, pp. 1–44, 2022.

[85] O. R. Gatla et al., "Towards robust file system checkers," *ACM Trans. Storage*, vol. 14, pp. 1–25, 2018.

[86] R. Han, D. Zhang, and M. Zheng, "Fingerprinting the checker policies of parallel file systems," in *Proc. IEEE/ACM 5th Int. Parallel Data Syst. Workshop*, 2020, pp. 46–51.

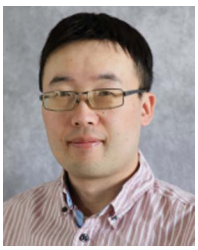
[87] D. Zhang, D. Dai, R. Han, and M. Zheng, "SentiLog: Anomaly detecting on parallel file systems via log-based sentiment analysis," in *Proc. 13th ACM Workshop Hot Top. Storage File Syst.*, 2021, pp. 86–93.

[88] E. Xu, M. Zheng, F. Qin, Y. Xu, and J. Wu, "Lessons and actions: What we learned from 10k SSD-related storage system failures," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 961–976.

[89] D. Huang, X. Zhang, W. Shi, M. Zheng, S. Jiang, and F. Qin, "LiU: Hiding disk access latency for HPC applications with a new SSD-enabled data layout," in *Proc. IEEE 21st Int. Symp. Modelling Anal. Simul. Comput. TeleCommun. Syst.*, 2013, pp. 111–120.

[90] D. Zhang and M. Zheng, "Benchmarking for observability: The case of diagnosing storage failures," *BenchCouncil Trans. Benchmarks, Standards Eval.*, vol. 1, 2021, Art. no. 100006.
[91] D. Zhang, C. Egersdoerfer, T. Mahmud, M. Zheng, and D. Dai, "Drill: Log-based anomaly detection for large-scale storage systems using source code analysis," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2023, pp. 189–199.
[92] E. Xu, M. Zheng, F. Qin, J. Wu, and Y. Xu, "Understanding SSD reliability in large-scale cloud systems," in *Proc. IEEE/ACM 3rd Int. Workshop Parallel Data Storage Data Intensive Scalable Comput. Syst.*, 2018, pp. 45–53.
[93] O. R. Gatla and M. Zheng, "Understanding the fault resilience of file system checkers," in *Proc. 9th USENIX Workshop Hot Top. Storage File Syst.*, 2017, pp. 7–7.
[94] Y. Shi, D. V. Murillo, S. Wang, J. Cao, and M. Zheng, "A command-level study of Linux kernel bugs," in *Proc. Int. Conf. Comput. Netw. Commun.*, 2017, pp. 798–802.
[95] J. Cao, S. Wang, D. Dai, M. Zheng, and Y. Chen, "A generic framework for testing parallel file systems," in *Proc. 1st Joint Int. Workshop Parallel Data Storage Data Intensive Scalable Comput. Syst.*, 2016, pp. 49–54.
[96] S. Wang, J. Cao, D. V. Murillo, Y. Shi, and M. Zheng, "Emulating realistic flash device errors with high fidelity," in *Proc. IEEE Int. Conf. Netw. Archit. Storage*, 2016, pp. 1–2.
[97] M. Zheng, V. T. Ravi, F. Qin, and G. Agrawal, "GRace: A low-overhead mechanism for detecting data races in GPU programs," in *Proc. 16th ACM Symp. Princ. Pract. Parallel Program.*, 2011, pp. 135–146.
[98] M. Zheng, V. T. Ravi, F. Qin, and G. Agrawal, "GMRace: Detecting data races in GPU programs via a low-overhead scheme," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 104–115, Jan. 2014.
[99] D. Dai, O. R. Gatla, and M. Zheng, "A performance study of lustre file system checker: Bottlenecks and potentials," in *Proc. 35th Symp. Mass Storage Syst. Technol.*, 2019, pp. 7–13.
[100] M. Zheng, J. Tucek, F. Qin, M. Lillibridge, B. W. Zhao, and E. S. Yang, "Reliability analysis of SSDs under power fault," *ACM Trans. Comput. Syst.*, vol. 34, pp. 1–28, 2016.
[101] M. Zheng, J. Tucek, F. Qin, and M. Lillibridge, "Understanding the robustness of SSDs under power fault," in *Proc. 11th USENIX Conf. File Storage Technol.*, 2013, pp. 271–284.
[102] S. Kamat, A. A. Raqibul Islam, M. Zheng, and D. Dai, "FaultyRank: A graph-based parallel file system checker," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2023, pp. 200–210.
[103] M. Zheng, V. T. Ravi, W. Ma, F. Qin, and G. Agrawal, "GMProf: A low-overhead, fine-grained profiling approach for GPU programs," in *Proc. 19th Int. Conf. High Perform. Comput.*, 2012, pp. 1–10.

Runzhou Han is currently working toward the PhD degree with the Department of Electrical and Computer Engineering at Iowa State University. His research interests include large scale storage systems, data provenance and serverless computing.

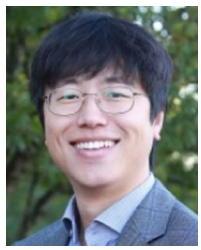


Mai Zheng received the PhD degree from The Ohio State University. He is an Associate Professor with the Department of Electrical and Computer Engineering, Iowa State University. He received the NSF CAREER award and a few Best Paper Nominations at FAST and HotStorage.

Suren Byna (Member, IEEE) is a professor with The Ohio State University. He was a senior scientist with the Scientific Data Division, Lawrence Berkeley National Lab (LBNL) before joining OSU. His research interests are in parallel I/O, data management systems for scientific data, etc.

Houjun Tang is currently a computer research scientist with the Scientific Data Management Group at Lawrence Berkeley National Lab (LBNL). His research interests include data management, storage systems, parallel I/O, and high performance computing.

Bin Dong is a research scientist with the Scientific Data Division, Lawrence Berkeley National Lab (LBNL). His research interests are in scientific data management and analysis, parallel computing, machine learning, scalable algorithms and data structures for indexing, etc.



Dong Dai received the PhD degree from the University of Science and Technology of China (USTC). He is an assistant professor in computer science with the Univ. of North Carolina at Charlotte. His research interests are in building intelligent infrastructure for high-performance data-intensive systems.

Yong Chen received the PhD degree from Michigan State University. He is a principal software engineer lead and architect with Samsung Electronics. He has contributed to releasing major software/hardware/systems in storage at Microsoft, Huawei, CNEX Labs and Stellus/Samsung.

Dongkyun Kim received the PhD degree from University California, San Diego. He is a senior staff engineer within Samsung Neural Processing Lab, overseeing HW and SW co-design of HPC computing. He also leads the technology intelligence group in Samsung Advanced Institute of Technology (SAIT).

Joseph Hassoun received the BS EE degree from University of Michigan and the MS EE degree from Stanford University. He is the director of the Neural Processing Lab at Samsung Electronics. He was awarded 29 patents in the fields of Computer Architecture, Deep Learning, and Circuit Design.

David Thorsley received the PhD degree in electrical engineering from the University of Michigan. He is a technical lead on the algorithms and machine learning group within the Samsung Neural Processing Lab. He has also published research in control theory, synthetic biology, and physiology.