PROV-IO: An I/O-Centric Provenance Framework for Scientific Data on HPC Systems

Runzhou Han
hanrz@iastate.edu
Iowa State University
Ames, Iowa, USA

Suren Byna
sbyna@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

Houjun Tang
htang4@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

Bin Dong
dbin@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

Mai Zheng
mai@iastate.edu
Iowa State University
Ames, Iowa, USA

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 three 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 provides 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.

CCS CONCEPTS

• **Computer systems organization** → **Parallel architectures**; • **Information systems** → **Data management systems**.

KEYWORDS

Data Provenance, Lineage, Scientific Data, Workflows, High Performance Computing, FAIR Principles, Explainability, Trustworthiness

ACM Reference Format:
Runzhou Han, Suren Byna, Houjun Tang, Bin Dong, and Mai Zheng. 2022. PROV-IO: An I/O-Centric Provenance Framework for Scientific Data on HPC Systems. In *Proceedings of the 31st Int'l Symposium on High-Performance Parallel and Distributed Computing (HPDC '22), June 27-July 1, 2022, Minneapolis, MN, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3502181.3531477

HPDC '22, June 27-July 1, 2022, Minneapolis, MN, USA

ACM ISBN 978-1-4503-9199-3/22/06.
https://doi.org/10.1145/3502181.3531477

1 INTRODUCTION

1.1 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.

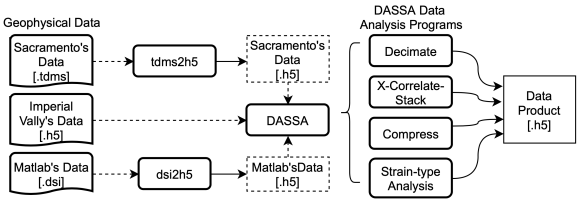

Figure 1: DASSA workflow. Solid arrows stand for write operation and dashed arrows stand for read operation.

As an example, Figure 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 different 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 I/O), which makes tracking and deriving the data provenance non-trivial. Moreover, as we will elaborate in Section §3, there are other diverse needs of provenance for different scientific workflows and data (e.g., I/O statistics, lineage of configurations). Such diversity, complexity, as well as the stringent performance requirement in HPC environments call for a comprehensive and practical solution beyond the state of the art.

1.2 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 multiple reasons. We summarize three main limitations below:

First, while the importance of provenance has been well recognized across communities in general (e.g., databases [6]–[10], operating systems (OS) [11], [12], eScience [13]–[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, 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 standards (e.g., W3C PROV [19]) are too vague to describe 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. Such ambiguity limits 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 the provenance APIs to track the desired information accordingly. Such labor-intensive and error-prone method hinders 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.

1.3 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 three 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]–[22], we introduce an I/O-centric provenance model dedicated for the HPC environments. The model enriches 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 I/O-centric 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.

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. We deploy PROV-IO on a representative supercomputer and evaluate it with realistic workflows. Our experiments show that PROV-IO incurs reasonable performance overhead and outperforms a state-of-the-art provenance product (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 three 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;
- 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.

1.4 Experimental Methodology & Artifact Availability

Experiments were performed on up to 64 compute nodes with Intel Xeon "Haswell" processors and with up to 4096 cores. The storage backend is a Lustre file system with typical configurations. We applied PROV-IO to three scientific workflows including DASSA [5], Top Reco [24], and a I/O-intensive application based on H5bench [25], which covers 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. 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 tool is open-source at https://github.com/hpc-io/prov-io.

1.5 Limitations of PROV-IO

The design of the PROV-IO tool is driven by the needs of the domain scientists using three 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.

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 [26]) 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 [27]) 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.

2 BACKGROUND

2.1 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 [28]. One key specification is PROV-DM, an extensible relational model which describes provenance information with a graph representation. As shown in Figure 2, a W3C provenance graph abstracts information into classes of *Entity*, *Activity*, *Agent*, and *Relation* between the first 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 follow the W3C PROV standard in the design of our PROV-IO model.

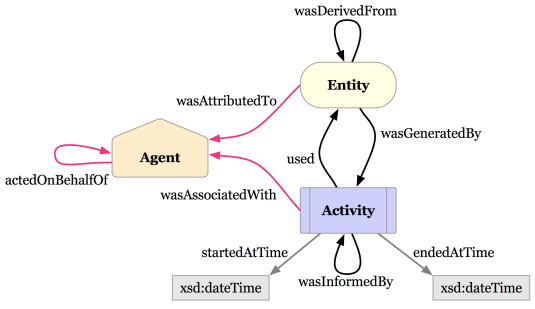

Figure 2: The W3C Provenance Model [28].

2.2 HPC I/O Libraries

I/O libraries (e.g., ADIOS [26], HDF5 [29], and NetCDF [30]) 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 [31]. 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.

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 [32]. 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.

3 CASE STUDIES

In this section, we discuss three 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.

3.1 Top Reco - Lineage of configurations

Workflow Description. Top Reco [24] 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 Figure 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 1, the Top Reco workflow uses the POSIX I/O interface, and involves multiple programs accessing multiple files.

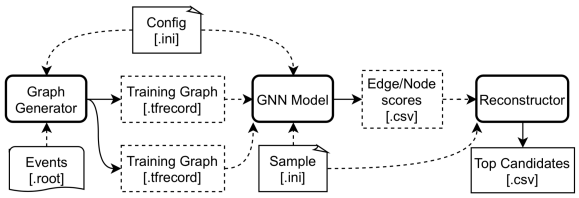

Figure 3: Top Reco workflow. Solid arrows stand for write operation and dashed arrows stand for read operation.

Provenance Need. In the Top Reco case, the domain scientists are interested in the impact of GNN configurations on the model

Table 1: Three 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

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 of the models.

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 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.

3.2 DASSA - Lineage of Data Products

Workflow Description. As mentioned in Section §1.1, DASSA [5] is a parallel storage and analysis framework for distributed acoustic sensing (DAS) applications. 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 Figure 1.

Provenance Need. As discussed in Section §1.1, 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.

3.3 H5bench - Data usage and I/O performance

Workflow Description. H5bench [25] is a parallel I/O benchmark suite for HDF5 [33] 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 1, the H5bench-based workflow mainly uses the HDF5 I/O interface, and involves multiple programs accessing a single file.

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 [34] and Recorder [35] 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.

3.4 Summary

By analyzing the three 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.

4 PROV-IO DESIGN

In this section, we introduce the design of PROV-IO. We focus on the provenance model (§4.1) and its system architecture (§4.2), which are two fundamental pillars of PROV-IO. We defer the implementation details to §5.

4.1 PROV-IO Model

Figure 4(a) shows an overview of the PROV-IO model, which is derived based on the W3C standard (§2.1) as well as the characteristics of typical workflows and the provenance needs of domain scientists (§3).

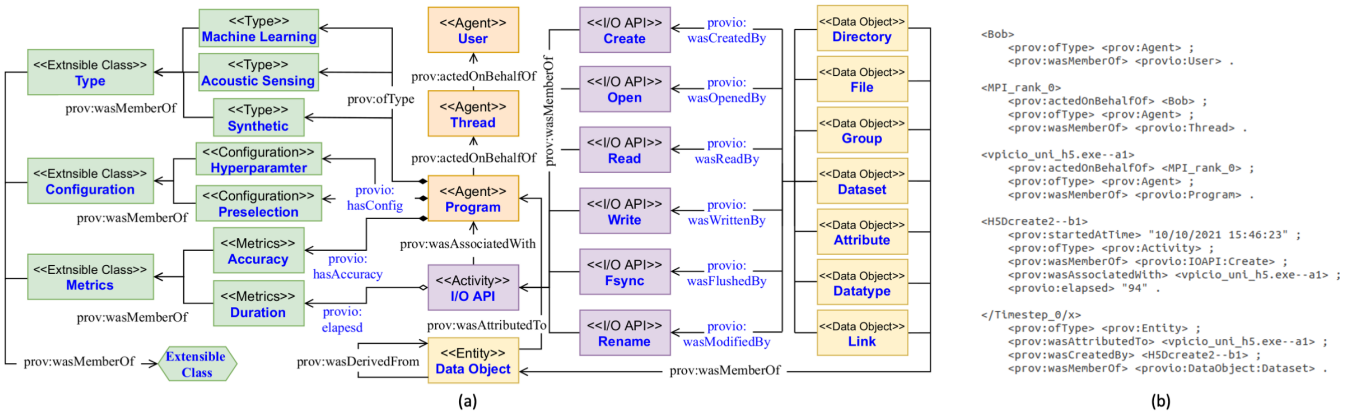

Figure 4: (a) PROV-IO Model Overview. The PROV-IO model classifies information into five super-classes: *Entity* (yellow boxes), *Activity* (purple boxes), *Agent* (orange boxes), *Extensible Class* (green boxes) and *Relation* (text on arrows). The new concepts introduced by PROV-IO are highlighted with blue color font. (b) A Provenance Snippet based on the PROV-IO Model.

Following the W3C specification, we classify information into five PROV-IO super-classes: *Entity* (yellow boxes in Figure 4(a)), *Activity* (purple boxes), *Agent* (orange boxes), *Extensible Class* (green boxes) and *Relation* (text on arrows). 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. We summarize the definitions of the sub-classes in Table 2 and highlight the main concepts added to each super-class as follows:

4.1.1 Entity. This PROV-IO super-class includes seven specific *Data Object* sub-classes (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 [36].

4.1.2 Activity. This super-class includes six specific *I/O API* sub-classes (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 [30]).

4.1.3 Agent. This super-class includes a set of sub-classes representing the operator of a series of activities, such as *User*, *Thread*, and *Program*. Note that the *Thread* sub-class can describe thread/process information (e.g., MPI rank) in multi-threaded programs which is critical in HPC environments.

4.1.4 Extensible class. This super-class contains properties pertained by entities, activities and agents. It is designed to be extensible because valuable information is often workflow-specific. We define three generic sub-classes (i.e., *Type*, *Configuration*, *Program*) to cover a variety of valuable information that cannot be described precisely in the native W3C specification (e.g., hyperparameters of ML models).

4.1.5 Relation. This super-class describes the diverse relations among other classes. We inherit the basic W3C provenance relations between *entity* & *entity* (prov:wasDerivedFrom), *entity* & *agent* (prov:wasAttributedTo), *activity* & *agent* (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).

To make the description more concrete, we show an example snippet of provenance captured by PROV-IO in Figure 4(b). The provenance snippet contains five records pertained by different subjects. Each subject can be an *Agent* (e.g., "Bob", "MPI_rank_0"), 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" on thread "MPI_rank_0", which was started by user "Bob").

4.2 PROV-IO Architecture

Figure 5 shows the architecture of the PROV-IO framework. Besides the PROV-IO model (yellow), the framework includes three major components: (1) provenance tracking (blue modules) which captures I/O operations from multiple I/O interfaces; (2) a provenance store (green) which persists captured provenance into RDF triples; (3) a user engine (red) for users to query and visualize provenance information. We introduce the design of these three major components one by one below.

Provenance Tracking. As shown in Figure 5, a scientific workflow is typically started on compute nodes by a user. The workflow may consist of several parallel applications with multiple threads

Table 2: Description of PROV-IO Model.

Super-class	Sub-class	Description
Entity	<<Data Object>> Directory	POSIX file system directory.
	<<Data Object>> File	POSIX file system file.
	<<Data Object>> Group	I/O library interior group structure (e.g., HDF5 group).
	<<Data Object>> Dataset	I/O library interior dataset structure (e.g. , HDF5 dataset).
	<<Data Object>> Attribute	POSIX Inode extended attribute and I/O library interior attribute structure (e.g., HDF5 attribute).
	<<Data Object>> Datatype	I/O library interior datatype structure (e.g., HDF5 datatype).
	<<Data Object>> Link	POSIX file system hard/soft link.
Activity	<<I/O API>> Create	POSIX syscall "open" and I/O library "Create" APIs (e.g., H5Acreate).
	<<I/O API>> Open	I/O library "Open" APIs (e.g., H5Aopen).
	<<I/O API>> Read	POSIX syscall "read" (and variants) and I/O library "Read" APIs (e.g., H5Aread).
	<<I/O API>> Write	POSIX syscall "write" (and variants) and I/O library "Write" APIs (e.g., H5Awrite).
	<<I/O API>> Fsync	POSIX syscall "fsync" (and variants) and I/O library "Flush" APIs (e.g., H5Flush).
	<<I/O API>> Rename	POSIX syscall "rename" (and variants) and I/O library "Rename" APIs.
Agent	User	Workflow user.
	Thread	Individual thread.
	Program	Program instance.
Extensible Class	Type	Type of a program/workflow (e.g., Machine Learning (Top Reco), Acoustic Sensing (DASSA), and Synthetic (H5bench workflow)).
	Configuration	Workflow configurations (e.g., hyperparameter in Top Reco).
	Metrics	Evaluation metrics of the workflow. E.g., model accuracy in Top Reco.
Relation	provio: wasCreatedBy	The relation between a <<I/O API>> Create and a <<Data Object>>.
	provio: wasOpenedBy	The relation between a <<I/O API>> Open and a <<Data Object>>.
	provio: wasReadBy	The relation between a <<I/O API>> Read and a <<Data Object>>.
	provio: wasWrittenBy	The relation between a <<I/O API>> Write and a <<Data Object>>.
	provio: wasFlushedBy	The relation between a <<I/O API>> Fsync and a <<Data Object>>.
	provio: wasModifiedBy	The relation between a <<I/O API>> Rename and a <<Data Object>>.

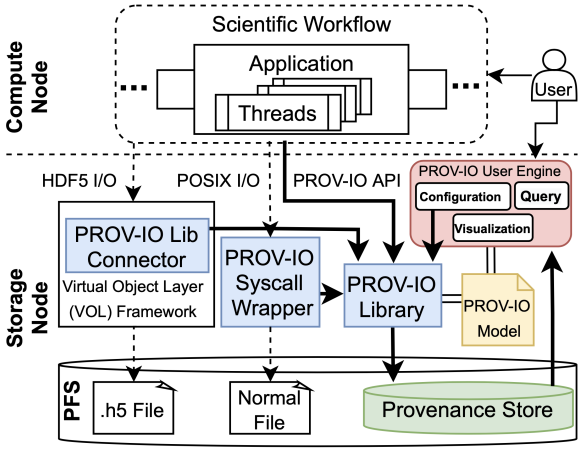

Figure 5: The Architecture of PROV-IO Framework. Beside the PROV-IO model (yellow), the framework includes three major components: provenance tracking (blue modules), a provenance store (green) and a user engine (red).

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 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 [37].

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.

5 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 PROV-IO Lib Connector in C and integrate it with the native HDF5 VOL-provenance connector, which follows a homomorphic design in which each HDF5 native I/O API has a counterpart API [32]. Upon each invocation of an HDF5 native API, the counterpart API adds the corresponding virtual data object to a linked list. PROV-IO Lib Connector 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 HDF5 VOL-provenance connector. *Entity* and *Activity* classes are tracked at each homomorphic API during the workflow runtime.

Similarly, to support POSIX I/O, we use GOTCHA [38] 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.

Provenance Store. The Provenance Store is implemented based on Redland `librdf` [39] 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 [40]–[44].

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 [45], ntriples [46], etc.). Redland `librdf` also supports the integration of multiple databases as the storage backend (e.g., BerkeleyDB, MySQL, PostgreSQL and 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 help performance because no extra inter-process communication or synchronization is needed during workflow execution, and the merging can be performed after workflow execution.

PROV-IO User Engine. Also, the engine supports querying RDF triples with SPARQL, which is a semantic query language to retrieve and manipulate data stored in RDF [47]. We use Python scripts as the SPARQL endpoint. Note that depending on different use case scenarios, the query can vary a lot, as will be demonstrated in Section §6.5. In the current prototype, we utilize Graphviz [48] for RDF graph visualization.

6 EVALUATION

In this section, we evaluate a prototype of the PROV-IO framework in representative HPC environments. We first introduce the experimental methodology (§6.1), and then evaluate PROV-IO from three perspectives including tracking performance (§6.2), storage requirement (§6.3), and query effectiveness for the end users (§6.5). We compare PROV-IO with a state-of-the-art provenance product (i.e., ProvLake [20]) in §6.4. Overall, our experimental results shows that PROV-IO's 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.

Table 3: The provenance needs and the information tracked by PROV-IO for three workflows.

Workflow	Provenance Need	Information Tracked
Top Reco (Python)	metadata version control & mapping	hyperparameter, preselection, training accuracy
DASSA (C++)	file lineage	program, I/O API, file
	dataset lineage	program, I/O API, dataset
	attribute lineage	program, I/O API, attr
H5bench (C)	scenario-1	I/O API
	scenario-2	I/O API, duration
	scenario-3	user, thread, program, file

6.1 Experimental Methodology

We have evaluated the PROV-IO framework on a state-of-the-art supercomputer. We do experiments with 64 Intel Xeon "Haswell" processor nodes and up to 4096 cores, unless otherwise specified. The storage backend is a Lustre file system with stripe count of 128 and stripe size of 16MB.

We apply PROV-IO to three representative workflows including Top Reco [24], DASSA [5], and an H5bench-based workflow [25]. As mentioned in §3, 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 3 and elaborate them in detail in the following subsections.

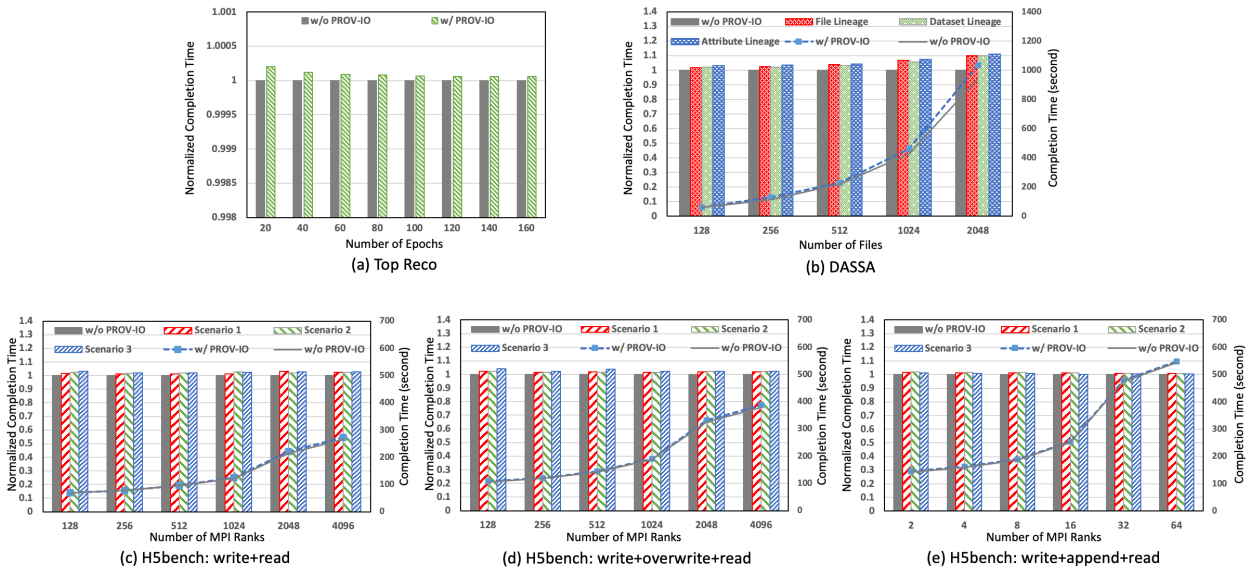

Figure 6: Performance of Provenance Tracking. (a) Top Reco. (b) DASSA (with File, Dataset, Attribute Lineages tracked). (c - e) H5bench-based workflow under three I/O patterns (i.e., write+read, write+overwrite+read, write+append+read)

6.2 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.

Figure 6(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). 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 3, PROV-IO tracks the information of user, program, file, dataset, or attribute for different lineage needs based on the PROV-IO model (§4.1). We follow a similar configuration as the domain scientists' by using 32 compute nodes and up to 2048 input files (1.35TB in total).

Figure 6(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 stands for 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 enabled under all usage 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 (ranging 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 based on different needs. As summarized in Table 3, *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*, *write-overwrite-read*, and *write-append-read*. 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

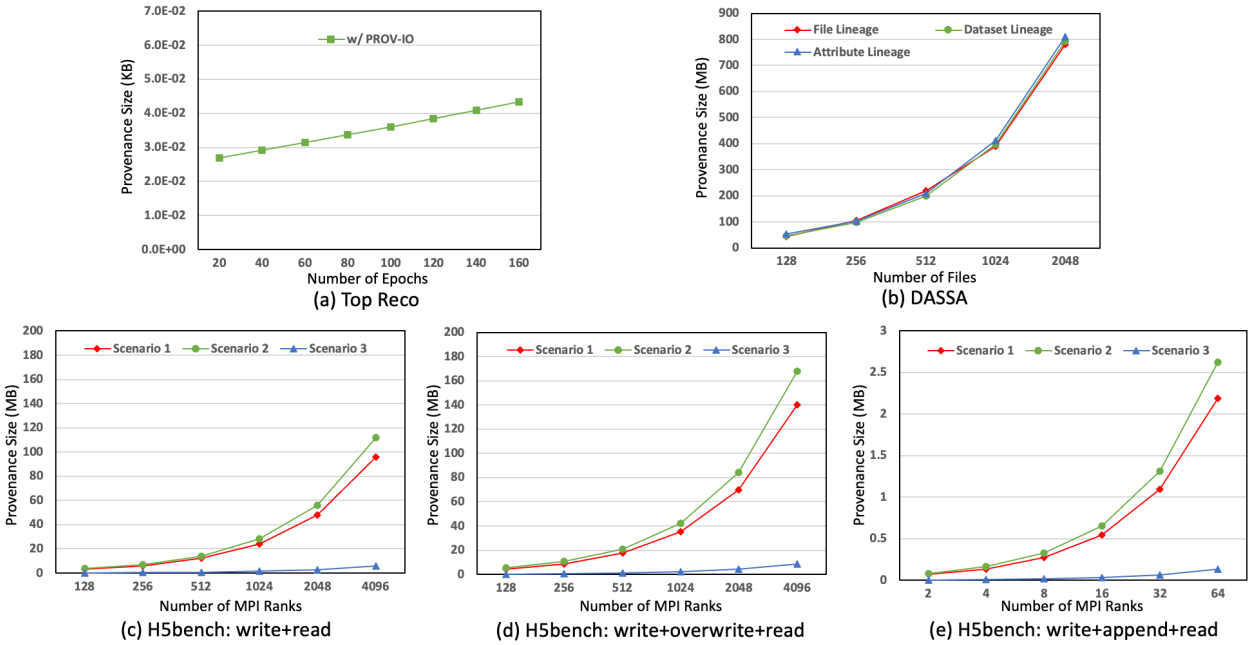

Figure 7: Storage of Provenance Tracking. (a) Top Reco. (b) DASSA (with File, Dataset, Attribute Lineages tracked). (c - e) H5bench-based workflow under three I/O patterns (i.e., write+read, write+overwrite+read, write+append+read)

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.

Figure 6 (c) (d) (e) show the tracking performance under three different I/O patterns (i.e., "write+read", "write+overwrite+read", "write+append+read") respectively. 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 3(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 (3.9TB data with 4096 MPI ranks). In particular, the PROV-IO overhead under the "write-append-read" I/O pattern (Figure 6 (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.

6.3 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 Figure 7.

Figure 7(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.

Figure 7(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.

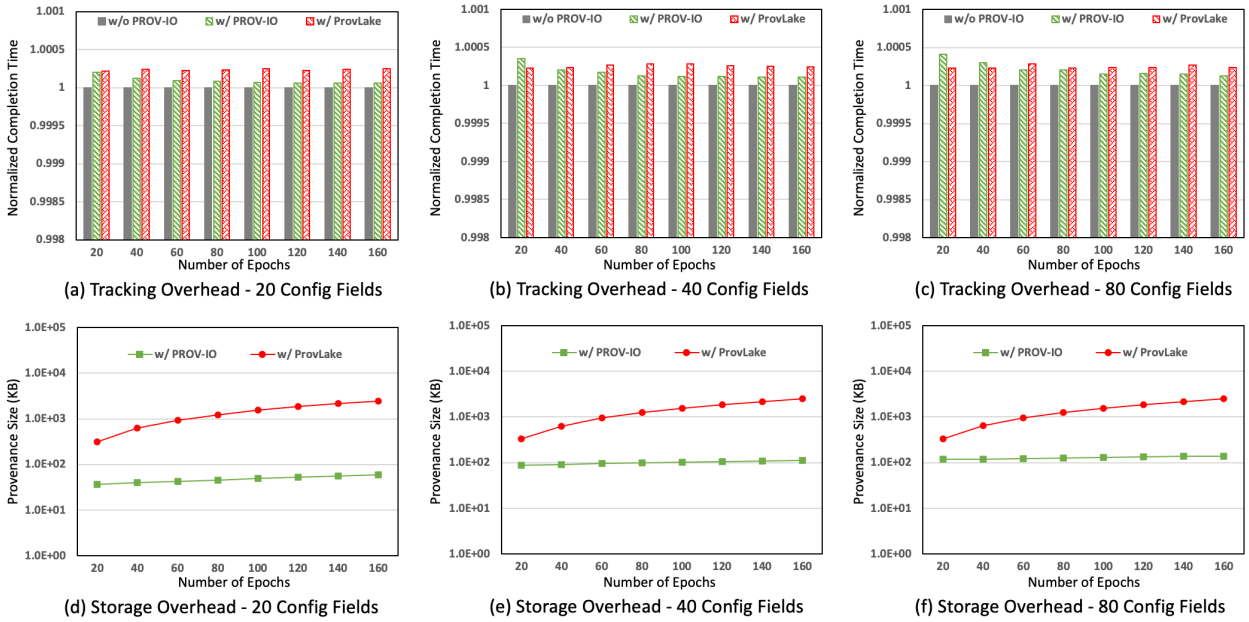

Figure 8: A performance comparison between PROV-IO and ProvLake.

Figure 7 (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 3). We can see the provenance size varies from a few KBs to 168 MBs. Among the three I/O patterns, "write+overwrite+read" has the highest storage overhead under usage scenario 2. This is because the pattern includes one more I/O application (i.e., overwrite) than "write+read" and has much more MPI processes contributing to provenance graph than "write+append+read". 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.

6.4 Comparison with Other Frameworks

In this section, we compare PROV-IO to state-of-the-art provenance systems. Table 4 shows the basic characteristics of Komadu [49], 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.

Table 4: 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

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 (§4.1). 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).

Figure 8(a),(b),(c) compares the provenance tracking performance of the two systems where y-axis is normalized completion time. Figure 8 (d),(e),(f) shows the storage overhead where y-axis is size in

Table 5: Example Queries. The diverse provenance needs can be satisfied by a few simple queries effectively.

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_1; 3: provio:wasReadBy ?IO_API.	3*N (where N is backward propagation steps)
H5bench	scenario-1	4: ?IO_API prov:wasMemberOf prov:Activity;	1
	scenario-2	5: ?IO_API prov:wasMemberOf prov:Activity; 6: provio:elapsed ?duration.	2
	scenario-3	7: file_a prov:wasAttributedTo ?program. 8: program_1 prov:actedOnBehalfOf ?thread. 9: thread_i prov:actedOnBehalfOf ?user.	3
Top Reco	metadata version control & mapping	10: ?configuration ns1:Version ?version; 11: provio:hasAccuracy ?accuracy.	2

KB. In all figures x-axis is the number of configurations. In (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 (d)(e)(f), green lines stand for PROV-IO provenance file size and red lines stand for ProvLake provenance file size.

As shown in Figure8(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 Figure8(d)(e)(f), PROV-IO always incurs less storage overhead, regardless of the number of configuration fields tracked. This is mainly because ProvLake has to track more irrelevant workflow information not needed in the use case.

6.5 Query Effectiveness

As mentioned in §5, PROV-IO supports provenance query with visualization. Table 5 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", we first query with keywords "decimate.h5 prov:wasAttributedTo ?program" to locate program "decimate", and then query "decimate"'s input file "WestSac.h5" with keywords "?file wasAttributedTo decimate". We can use similar queries to locate earlier predecessors (e.g., "WestSac.tdms"). As summarized in Table 5, for each backward step, we only need three query statements. Figure 9 shows the visualization of this example, which follows the PROV-IO provenance model (§4.1) 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 §6.2) 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.

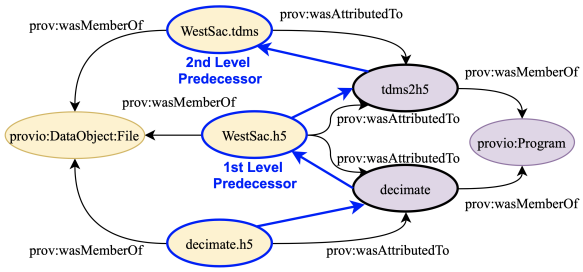

Figure 9: An Example of DASSA Data Lineage by PROV-IO. The graph follows the PROV-IO model; the data lineage is highlighted in blue.

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 the PROVI-IO model as well as the entire framework is effective for scientific data on HPC systems.

7 RELATED WORK

In this section, we discuss provenance-related work and tools that have not been covered sufficiently in the previous sections.

Database Provenance. Historically, provenance has been well studied in databases to understand the causal relationship between materialized views and table updates [6], [50]. The concept has also been extended to other usages [8], [51]. In general, database provenance may leverage the well-defined relational model and the relatively strict transformations to capture precise provenance within the system [52], which is not applicable for general software. On the other hand, some query optimizations (e.g., provenance reduction [53]) could potentially be applied to PROV-IO. Therefore, PROV-IO and existing database tools are complementary.

OS-Level Provenance. Great efforts have also been made to capture provenance at the operating system (OS) level [11], [12], [37]. 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 (§4.1). 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 specifications.

(2) *System Architecture*: PROV-IO is a user-level solution designed for the HPC environment (§4.2). 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).

(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 [47], 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 tools.

Workflow & Application Provenance. Provenance models or systems for workflows and/or applications have also been explored [18], [20], [21], [54]. 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 [55]–[57].

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 [58]. 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 Tools. In addition, great efforts have been made to manage workflows [59], [60] or log I/O events for various purposes [34], [35], [61]–[75]. 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 tracking or querying rich provenance (e.g., various relations defined in W3C PROV-DM [28]); (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 workflow environments; (4) no programmable interface to specify customized provenance needs.

8 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. We believe that PROV-IO represents a promising direction toward ensuring the rigorousness and trustworthiness of scientific data management.

In the future, we will address the limitations mentioned in §1.5. 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.

9 ACKNOWLEDGMENTS

The authors would like to thank Yogesh Simmhan (our shepherd) and the anonymous reviewers for their insightful feedback. We also thank Xiangyang Ju for providing the Top Reco workflow. This work was supported in part by NSF under grants CNS-1855565, CCF-1853714, CCF-1910747 and CNS-1943204. 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 Contract No. 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 *et al.*, "The fourth paradigm: Data-intensive scientific discovery [point of view]," *Proceedings of the IEEE*, vol. 99, no. 8, pp. 1334–1337, 2011. DOI: 10.1109/JPROC.2011.2155130.

[2] M. D. Wilkinson *et al.*, *The fair guiding principles for scientific data management and stewardship*, 2016.

[3] *Fair principles.* [Online]. Available: https://www.go-fair.org/fair-principles/ (visited on 10/04/2021).

[4] M. Milton *et al.*, "Trustworthy data underpin reproducible research," *Nature Physics*, vol. 16, pp. 117–119, Feb. 2020. DOI: 10.1038/s41567-019-0780-5.

[5] B. Dong *et al.*, "DASSA: Parallel DAS Data Storage and Analysis for Subsurface Event Detection," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS'20)*, 2020.

[6] P. Buneman *et al.*, "Why and where: A characterization of data provenance," in *Proceedings of the 8th International Conference on Database Theory (ICDT'01)*, 2001.

[7] X. Niu *et al.*, "Interoperability for provenance-aware databases using PROV and JSON," in *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP'15)*, 2015.

[8] P. Senellart, "Provenance and probabilities in relational databases," *SIGMOD Rec.*, vol. 46, no. 4, pp. 5–15, Feb. 2018, ISSN: 0163-5808. DOI: 10.1145/3186549.3186551.

[9] Z. Miao *et al.*, "Going beyond provenance: Explaining query answers with pattern-based counterbalances," in *Proceedings of the 2019 International Conference on Management of Data (SIGMOD'19)*, 2019.

[10] Z. Wang *et al.*, "A provenance storage method based on parallel database," in *2015 2nd International Conference on Information Science and Control Engineering (ICISCE'15)*, 2015.

[11] K.-K. Muniswamy-Reddy *et al.*, "Provenance-aware storage systems," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference (ATC'06)*, 2006.

[12] K.-K. Muniswamy-Reddy *et al.*, "Layering in provenance systems," in *Proceedings of the 2009 Conference on USENIX Annual Technical Conference (ATC'09)*, 2009.

[13] E. Jandre *et al.*, "Provenance in collaborative in silico scientific research: A survey," *SIGMOD Rec.*, vol. 49, no. 2, pp. 36–51, Dec. 2020, ISSN: 0163-5808. DOI: 10.1145/3442322.3442329.

[14] Y. L. Simmhan *et al.*, "A survey of data provenance in e-science," *SIGMOD Rec.*, vol. 34, no. 3, pp. 31–36, Sep. 2005, ISSN: 0163-5808. DOI: 10.1145/1084805.1084812.

[15] S. B. Davidson *et al.*, "Provenance and scientific workflows: Challenges and opportunities," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, 2008.

[16] I. Suriarachchi *et al.*, "Big provenance stream processing for data intensive computations," in *2018 IEEE 14th International Conference on eScience (eScience'18)*, 2018.

[17] R. Souza *et al.*, "Efficient runtime capture of multiworkflow data using provenance," in *2019 15th International Conference on eScience (eScience'19)*, 2019.

[18] R. Souza *et al.*, "Provenance data in the machine learning lifecycle in computational science and engineering," in *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS'19)*, 2019.

[19] *The prov data model - w3c.* [Online]. Available: https://www.w3.org/TR/prov-overview/ (visited on 10/04/2021).

[20] L. Azevedo *et al.*, "Experiencing provlake to manage the data lineage of ai workflows," in *Meeting in Innovation in Information Systems (EISI) in Brazilian Symposium in Information Systems (SBSI'20)*, 2020.

[21] Y. L. Simmhan *et al.*, "A framework for collecting provenance in data-centric scientific workflows," in *2006 IEEE International Conference on Web Services (ICWS'06)*, 2006.

[22] B. Howe *et al.*, "End-to-end escience: Integrating workflow, query, visualization, and provenance at an ocean observatory," in *2008 IEEE Fourth International Conference on eScience (eScience'08)*, 2008.

[23] *Resource description framework.* [Online]. Available: https://www.w3.org/RDF/ (visited on 10/04/2021).

[24] X. Allison *et al.*, *A graph neural network-based top quark reconstruction package.* [Online]. Available: https://indico.cern.ch/event/932415/contributions/3918265/attachments/2086561/3505362/GNN_Top_Reco_-_Allison_Xu.pdf (visited on 10/04/2021).

[25] *H5bench.* [Online]. Available: https://www.hdfgroup.org/solutions/hdf5/ (visited on 10/04/2021).

[26] *Adios.* [Online]. Available: https://www.olcf.ornl.gov/center-projects/adios/.

[27] A. Bates *et al.*, "Trustworthy whole-system provenance for the linux kernel," in *Proceedings of the 24th USENIX Conference on Security Symposium (Security'15)*, 2015.

[28] P. Missier *et al.*, "The w3c prov family of specifications for modelling provenance metadata," in *Proceedings of the 16th International Conference on Extending Database Technology (EDBT'13)*, 2013.

[29] *Hdf5.* [Online]. Available: https://https://www.hdfgroup.org/solutions/hdf5/ (visited on 10/04/2021).

[30] *Netcdf.* [Online]. Available: https://www.unidata.ucar.edu/software/netcdf/ (visited on 10/04/2021).

[31] *Automatic library tracking database at nersc.* [Online]. Available: https://www.nersc.gov/assets/altdatNERSC.pdf (visited on 10/04/2021).

[32] T. Li *et al.*, "H5prov: I/o performance analysis of science applications using hdf5 file-level provenance," in *Cray User Group (CUG'19)*, 2019.

[33] M. Folk *et al.*, "An overview of the hdf5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases (AD'11)*, 2011.

[34] *Darshan, hpc i/o characterization tool.* [Online]. Available: https://www.mcs.anl.gov/research/projects/darshan/ (visited on 10/04/2021).

[35] S. Yellapragada *et al.*, "Verifying io synchronization from mpi traces," in *2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW'21)*, 2021.

[36] *Ext4.* [Online]. Available: https://ext4.wiki.kernel.org/index.php/Main_Page (visited on 10/04/2021).

[37] A. Gehani *et al.*, "Spade: Support for provenance auditing in distributed environments," in *Proceedings of the 13th International Middleware Conference (Middleware'12)*, 2012.

[38] *Gotcha v1.0.2.* [Online]. Available: https://github.com/LLNL/GOTCHA (visited on 10/04/2021).

[39] *Redland rdf.* [Online]. Available: https://librdf.org (visited on 10/04/2021).

[40] *Apache jena.* [Online]. Available: https://jena.apache.org (visited on 10/04/2021).

[41] *Neo4j.* [Online]. Available: https://neo4j.com (visited on 10/04/2021).

[42] *Blazegraph.* [Online]. Available: https://blazegraph.com (visited on 10/04/2021).

[43] *Apache rya.* [Online]. Available: https://rya.apache.org (visited on 10/04/2021).

[44] *Anzographdb.* [Online]. Available: https://cambridgesemantics.com/anzograph/ (visited on 10/04/2021).

[45] *Terse rdf triple language.* [Online]. Available: https://www.w3.org/TR/turtle/ (visited on 10/04/2021).

[46] *N-triples.* [Online]. Available: https://www.w3.org/TR/n-triples/ (visited on 10/04/2021).

[47] *Sparql query language for rdf.* [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/ (visited on 10/04/2021).

[48] *Graphviz.* [Online]. Available: https://graphviz.org (visited on 10/04/2021).

[49] I. Suriarachchi *et al.*, "Komadu: A capture and visualization system for scientific data provenance," *Journal of Open Research Software*, vol. 3, Mar. 2015. DOI: 10.5334/jors.bq.

[50] Y. Cui *et al.*, "Tracing the lineage of view data in a warehousing environment," *ACM Trans. Database Syst.*, vol. 25, no. 2, pp. 179–227, Jun. 2000, ISSN: 0362-5915. DOI: 10.1145/357775.357777.

[51] J. Widom, "Trio: A system for integrated management of data, accuracy, and lineage," in *2nd Biennial Conference on Innovative Data Systems Research (CIDR'05)*, 2005.

[52] L. Carata *et al.*, "A primer on provenance," *Commun. ACM*, vol. 57, no. 5, pp. 52–60, May 2014, ISSN: 0001-0782. DOI: 10.1145/2596628.

[53] D. Deutch *et al.*, "Hypothetical reasoning via provenance abstraction," in *Proceedings of the 2019 International Conference on Management of Data (SIGMOD'19)*, 2019.

[54] Q. Zhou *et al.*, "Study in usefulness of middleware-only provenance," in *2014 IEEE 10th International Conference on eScience (eScience'14)*, 2014.

[55] *Braid-db.* [Online]. Available: https://github.com/ANL-Braid/DB/.

[56] *Chimbuko.* [Online]. Available: https://github.com/CODARcode/Chimbuko (visited on 10/04/2021).

[57] J. Logan *et al.*, "A vision for managing extreme-scale data hoards," in *IEEE 39th International Conference on Distributed Computing Systems (ICDCS'19)*, 2019.

[58] P. Alvaro *et al.*, "Lineage-driven fault injection," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*, 2015.

[59] *Apache taverna.* [Online]. Available: https://incubator.apache.org/projects/taverna.html (visited on 10/04/2021).

[60] *Effis.* [Online]. Available: https://github.com/wdmapp/effis (visited on 10/04/2021).

[61] M. Zheng *et al.*, "Torturing Databases for Fun and Profit," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[62] J. Cao *et al.*, "A generic framework for testing parallel file systems," in *2016 1st Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS)*, 2016.

[63] J. Cao *et al.*, "PFault: A general framework for analyzing the reliability of high-performance parallel file systems," in *Proceedings of the 2018 International Conference on Supercomputing (ICS)*, 2018, pp. 1–11. DOI: 10.1145/3205289.3205302.

[64] R. Han *et al.*, "A study of failure recovery and logging of high-performance parallel file systems," *ACM Transactions on Storage (TOS)*, 2021. DOI: 10.1145/3483447.

[65] O. R. Gatla *et al.*, "Understanding the fault resilience of file system checkers," in *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2017.

[66] O. R. Gatla *et al.*, "Towards robust file system checkers," in *16th USENIX Conference on File and Storage Technologies (FAST)*, 2018.

[67] O. R. Gatla *et al.*, "Towards robust file system checkers," *ACM Transactions on Storage (TOS)*, vol. 14, no. 4, pp. 1–25, 2018. DOI: 10.1145/3281031.

[68] D. Zhang *et al.*, "Sentilog: Anomaly detecting on parallel file systems via log-based sentiment analysis," in *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2021.

[69] D. Dai *et al.*, "A performance study of lustre file system checker: Bottlenecks and potentials," in *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, 2019.

[70] E. Xu *et al.*, "Understanding ssd reliability in large-scale cloud systems," in *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, 2018.

[71] E. Xu *et al.*, "Lessons and actions: What we learned from 10k {ssd-related} storage system failures," in *2019 USENIX Annual Technical Conference (USENIX ATC)*, 2019.

[72] Y. Shi *et al.*, "A Command-Level Study of Linux Kernel Bugs," in *2017 International Conference on Computing, Networking and Communications (ICNC)*, 2017.

[73] M. Zheng *et al.*, "Gmprof: A low-overhead, fine-grained profiling approach for GPU programs," in *19th International Conference on High Performance Computing (HiPC)*, 2012. DOI: 10.1109/HiPC.2012.6507475.

[74] D. Huang *et al.*, "LiU: Hiding disk access latency for HPC applications with a new SSD-enabled data layout," in *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2013.

[75] D. Zhang *et al.*, "Benchmarking for Observability: The Case of Diagnosing Storage Failures," *BenchCouncil Transactions on Benchmarks, Standards and Evaluations (TBench)*, vol. 1, no. 1, 2021. DOI: 10.1016/j.tbench.2021.100006.