Building the I (Interoperability) of FAIR for Performance Reproducibility of Large-Scale Composable Workflows in RECUP

Bogdan Nicolae, Tanzima Islam, Robert Ross, Huub van Dam, Kevin Assogba, Polina Shpilker, Mikhail Titov, Matteo Turilli, Tianle Wang, Ozgur Kilic, et al.

Building the I (Interoperability) of FAIR for performance reproducibility of large-scale composable workflows in RECUP

Bogdan Nicolae[3], Tanzima Z. Islam[2], Robert Ross[3], Huub Van Dam[1], Kevin Assogba[5], Polina Shpilker[6]
Mikhail Titov[1], Matteo Turilli[1], Tianle Wang[1], Ozgur O. Kilic[1], Shantenu Jha[1,4], Line C. Pouchard[1]
[1]*Brookhaven National Laboratory,* [2]*Texas State University,* [3]*Argonne National Laboratory*
[4]*Rutgers University,* [5]*Rochester Institute of Technology,* [6]*Tufts University*
Email: {bnicolae,rross}@anl.gov, {hvandam,titov,mturillit,twang3,okilic,shantenu,pouchard}@bnl.gov,
tanzima@txstate.edu, kta7930@rit.edu, polina.shpilker@tufts.edu

***Abstract*—Scientific computing communities increasingly run their experiments using complex data- and compute-intensive workflows that utilize distributed and heterogeneous architectures targeting numerical simulations and machine learning, often executed on the Department of Energy Leadership Computing Facilities (LCFs). We argue that a principled, systematic approach to implementing FAIR principles at scale, including fine-grained metadata extraction and organization, can help with the numerous challenges to performance reproducibility posed by such workflows. We extract workflow patterns, propose a set of tools to manage the entire life cycle of performance metadata, and aggregate them in an HPC-ready framework for reproducibility (RECUP). We describe the challenges in making these tools interoperable, preliminary work, and lessons learned from this experiment.**

***Index Terms*—High performance computing, HPC, performance reproducibility, workflow execution patterns, workflow execution provenance, metadata capture, research software engineering, FAIR4RS, FAIR4HPC, RO-Crate.**

I. Introduction

Scientific computing communities increasingly run their experiments using complex data- and compute-intensive workflows that sit at the intersection of high performance computing (HPC), big data analytics, and artificial intelligence (AI). Such converged workflows often comprise a variety of patterns: ensembles of simulations with tasks executed in parallel, workflows that combine simulations with analytics and/or machine learning models responsible to act as surrogates and/or steer the simulation, data-driven workflows orchestrating tasks and their dependencies [1]. The workflows can be launched from simple scripts, from workflow management systems (WMS) representing execution in directed acyclic graphs, and from composable workflow building blocks. They often need to run at scale on large data centers and supercomputing infrastructures, such as US DOE Leadership Computing Facilities (LCFs) that include different types of accelerators, different types of storage hierarchies and data management capabilities, as well as different modes of operation (e.g., batch jobs combined with on-demand jobs).

The extreme heterogeneity from all perspectives (different types of tasks, patterns, infrastructure, job scheduling) poses numerous challenges to reproducibility [2], [3]. We cannot simply rely on packaging the application code into a container to achieve reproducibility [4], [5]. In addition we need to complement a similar execution environment with other aspects such as: the original input data, the workflow scripts and/or the definition of the workflow graph, and additional constraints if applicable (e.g., acceptable results as needed in the reproducibility of machine learning tasks [6], [7]). In this paper, we focus on performance reproducibility, i.e., the minimal run-to-run variation across multiple runs of the same application using a consistent configuration as described above. Performance reproducibility differs from performance portability as the latter pertains to achieving similar performance and scaling across systems, while the former indicates no variation across runs of the same application on an identical system. Specifically, we focus on the gaps in performance data and metadata collection that facilitates performance reproducibility.

The Findable, Accessible, Interoperable, Re-usable (FAIR) principles for data and software can be useful enablers for the reproducibility of performance and that of scientific results based on re-use. FAIR principles are under-used by the HPC and data-intensive communities who have been slow to adopt them [8], although some emerging efforts discussed in Section II are under way. Many pieces needed for reproducing either performance or results are missing (not findable); code is compiled with older versions of libraries that are no longer available on managed systems or the systems themselves have been de-commissioned (not accessible). Software modules require additional features to extract required results for performance reproducibility (not interoperable), and may not sufficiently characterize provenance (not re-usable). This is due in part to the complexity of workflow life cycles, the numerous workflow management systems available, the lack of integration of FAIR within existing technologies, the specificity of managed systems that include rapidly evolving architectures and software stacks, and execution models that require resource managers and batch schedulers.

The FAIR principles and subsequent research do not provide enough conceptual details on how to apply them to these HPC- and data-intensive environments and scientific practices. Numerous challenges emerge for scientists attempting to publish FAIR datasets and software created on such systems for the purpose of re-use and reproducibility, e.g. what data to publish and where due to sizes, how to reconstruct provenance graphs if they are not directly available, what minimal amount of metadata is needed to guarantee a certain level of reproducibility, etc. [9].

This work-in-progress paper describes our attempts, successes, and continuous challenges in enabling FAIR semantics, in particular with respect to the metadata that will influence performance reproducibility. Our position is that performance metadata cannot simply be collected and organized in a standardized format. Instead, it is necessary to manage its entire life-cycle in a scalable, low-overhead fashion. Our first contribution is to envision this life-cycle as a pipeline that begins with the application acting as a producer of performance metadata that needs to be collected with minimal overhead in a scalable fashion (in order to minimize the influence on the application runtime). The pipeline ends with an analytics code that acts (either offline or in real-time) as a consumer of the performance metadata to identify any potential divergences between multiple runs and their root cause. In between, there are multiple intermediate stages concerned with metadata curation, aggregation, indexing and query optimization, modular representation. While there are existing techniques and HPC tools to implement each of these stages, they were not designed to be FAIR and interoperable. Thus, our second contribution focuses on how bring together such techniques and HPC tools in order to implement the life-cycle as a pipeline. Specifically:

- We propose a set of tools and an associated architecture that is responsible to manage the entire life-cycle of the metadata needed to enable performance reproducibility as a pipeline. In particular, we insist on the importance of: (1) collection and curation; (2) aggregation and streaming; (3) indexing and query optimization (Section III-A).
- We discuss the challenges and preliminary work in making these tools interoperable. In particular, we highlight the importance of using lightweight representations of the performance metadata such that we enable comprehensive collection without sacrificing performance and scalability due to excessive overhead of metadata conversion and/or revisiting, or an explosion of storage space/memory utilization. Furthermore, we highlight the importance of preserving the performance metatdata in a standardized format (RO-Crate) in order to enable FAIR-ness (Section III-B – Section III-G).
- We summarize a series of general observations, lessons learned, and future opportunities in enabling interoperability of components and tools for performance reproducibility (Section IV).

II. Related Work

FAIR has been rapidly embraced by research data management communities that provide numerous successful examples of how to apply FAIR to ensure the reproducibility of published papers and emphasize data [10]. In [11] a property graph approach is explored for understanding the relationships between users, jobs, and datasets. SoMeta [12] is a scalable and decentralized metadata management approach for object-centric storage in HPC systems that provides an advantage over self-describing formats (such as HDF5, ADIOS, or PnetCDF) to include a scalable flat namespace and a tagging approach for extensible, user-defined metadata. The BRAID-DB Provenance Engine [13], designed for ML applications, records what goes into model training, including external data, simulations, and structures of learning and analysis activity. [14] and [15] argue for the automation of FAIR workflows in terms of composition of executable software steps, provenance extraction, and a common workflow language. Recording facts about runtime execution that can be inspected and updated enables provenance information focused on tracing dependencies and verifying validity.

Understanding performance reproducibility is an important precursor to developing adaptive system software that involves management of different heterogeneous hardware components and diverse resources, such as power, network, and I/O. Existing research has examined performance variation resulting from network interference [16], I/O congestion [17], and power-constrained scenarios that bring out chip-level manufacturing differences [18]. Predicting user estimates of the execution time of jobs to ensure they are not killed prematurely due to underestimation also has been studied [19]. However, all of these cited studies focus on metadata from only one system layer, e.g., either network or I/O or hardware. When there are cross-layer performance considerations, then finding underlying causes of performance perturbation is non trivial [20].

III. RECUP: A comprehensive performance reproducibility framework

A. Overview

As mentioned in Section I, modern scientific application workflow are complex and comprise a mix of HPC simulations, big data analytics and AI tasks. This can lead to a variety of patterns and task dependencies. In this regard, we observe several common aspects: (1) the workflows are made of multi-producer, multi-consumer tasks whose dependencies form complex directed acyclic graphs (DAGs); (2) each task may itself be complex and comprise a large number of coupled processes (e.g., a distributed simulation deployed on a large number of MPI ranks); (3) there may be significant fluctuations of performance metrics both in time and across task processes, which requires detailed metadata captured at fine granularity to understand root cause.

As a consequence, the first step in enabling performance reproducibility is the ability to capture a diverse set of performance metrics at fine granularity at scale. To this end, we identified four important types of metadata: (1) specifications of the DAG that describe the workflow behavior (which can be either statically defined before the workflow execution or dynamically constructed during runtime) and that can be used to identify the relationships between the tasks; (2) information about the runtime of each task (e.g., start timestamp, finish timestamp, duration of intermediate stages such as iterations, resource utilization over time, etc.); (3) types and durations, throughputs and latencies of I/O operations (necessary for the tasks to communicate inputs/outputs with each other); (4) anomalies that require more detailed metadata (as opposed to the normal mode of operation that can be characterized using much smaller metadata summaries).

To collect the four types of metadata introduced above, we propose to leverage several tools:

- **RADICAL Cybertools (RCT)** [21], which enables users to deploy workflows on HPC platforms and is responsible to manage the scheduling and execution of the workflow tasks on top of traditional HPC batch jobs. Of particular interest is the RADICAL-EnsembleToolkit (EnTK) component, which is responsible to provision the HPC resources by submitting HPC jobs, scheduling of

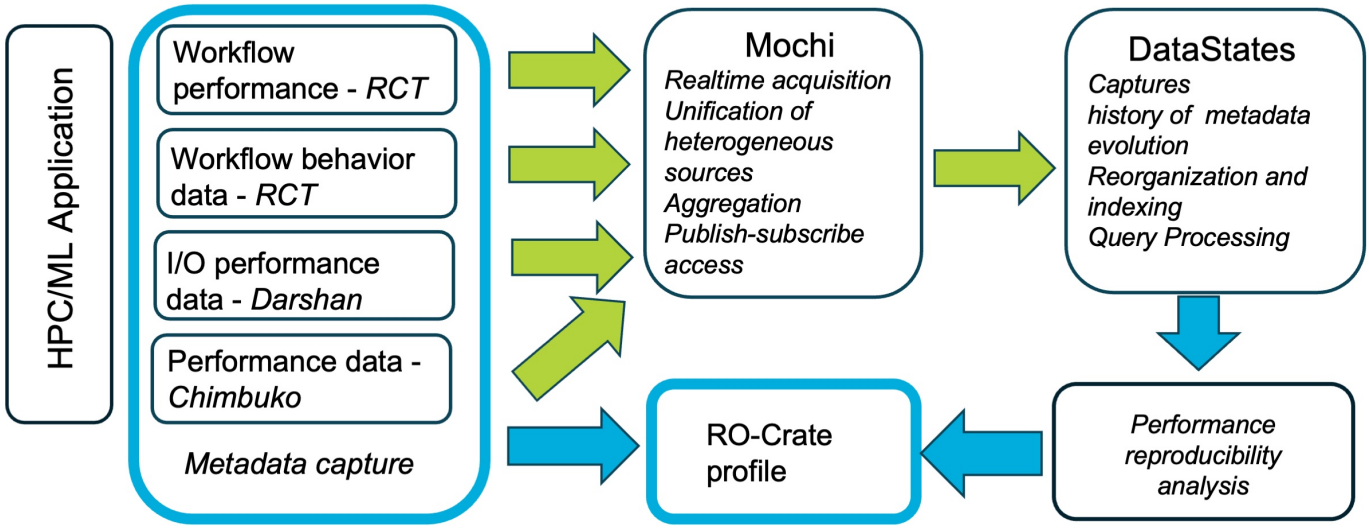

Fig. 1. Architecture of RECUP: the basic building blocks and their role in the management of the life-cycle of metadata in order to facilitate efficient performance reproducibility analytics.

computing tasks, and to monitor and log performance metadata for the tasks. The latter capability presents an opportunity to extend existing work to satisfy (1) and (2), which is the reason why we chose RCT.

- **Darshan** [22] is a scalable HPC I/O characterization tool used by many supercomputing infrastructures and large data centers that feature parallel file systems. It transparently monitors POSIX operations related to file I/O (open, read, write, close) and filesystem metadata operations (enumerate files, move/remove files/directories, etc.). In addition to retaining records of individual operations, it also generates statistics (number of operations for each type, duration/latency, etc.). Since a large number of tasks in a workflow use files to communicate intermediate results, Darshan is the ideal tool to satisfy (3).
- **Chimbuko** [23] is a tool for capturing, reducing, and visualizing in situ performance outliers and anomalies for extreme-scale workflows, with their call stack. Chimbuko provides detailed performance measurements at a trace level and can be easily extended to satisfy (4). Given that workflows comprise a large number of tasks, each of which can comprise a large number of processes that needs to be monitored at fine granularity, Chimbuko avoids an explosion of metadata by zooming on anomalies and retaining summaries for the normal mode of operation, all stored in its provenance database.

Capturing the relevant performance metadata at fine granularity from various sources is not enough to facilitate reproducibility analytics, because the metadata needs to be repeatedly revisited and analyzed from multiple perspectives. If the metadata were preserved in a raw form, this would lead to significant performance and scalability bottlenecks, or even would become unfeasible due to resource constraints (e.g. not enough memory).

As a consequence, the second step is to reorganize the metadata to alleviate performance, scalability and resource utilization bottlenecks. Specifically, there are two important aspects to consider: (1) curation, aggregation and streaming; (2) indexing and query optimization. With respect to (1), some of the performance metadata may be incorrect or irrelevant, in which case it can be discarded, or statistical properties and features are enough to describe it, in which case it can be aggregated. Notably, in a majority of scenarios, it is not necessary to wait for the collection of all metadata in order to perform these operations, which is why we propose to perform the operations on-the-fly using data streams. With respect to (2), since we repeatedly revisit the data from different perspectives, indexing and query optimization is needed in order to further alleviate the performance and scalability bottlenecks.

To this end, we leverage two additional tools:

- **Mochi** [24], a framework is designed for the rapid development of services for use in HPC platforms that includes composable building blocks and services such as RPC, unified key-value store access, group management, etc. Notably, data streaming services similar to Kafka are in developement, which would enable us to address (1).
- **DataStates** [25], a data management framework specifically designed to capture the evolution of intermediate data into a searchable lineage. Notably, it features efficient techniques for versioning and revision control with efficient navigation and search queries, which can be adapted to address (2).

Our overall vision is to extend and combine these tools such that we can manage the entire life-cycle of the performance metadata: starting with the collection from different sources to curation, aggregation, indexing, query optimization and finally the consumption by the reproducibility analytics. The architecture of our proposal is depicted in Figure 1.

The main challenge in realizing this vision lies in the fact that the pre-existing tools we propose to leverage were not initially designed to work together; Chimbuko and Darshan produce heterogeneous performance data and metadata at large scales, while EnTK only produces performance data for its own behavior (based on tracing start/stop events for EnTK components as well for handled computing tasks). Both Mochi and DataStates need to be adapted to support streams of timestamped events representing the performance metadata in order to be able to talk to each other. Thus, we need to design

connectors that facilitate interoperability between the tools and allow us to seamlessly compose them at runtime. For the rest of this section, we discuss the challenges and our preliminary work in facilitating interoperability.

B. Chimbuko + RCT

Several additional features to the RCT stack were needed to enable managing of external tools running as supplementary services, e.g., tools to continuously extract performance data and metadata, such as the Chimbuko performance analysis tool. The RCT stack (both RADICAL-Pilot and EnTK) was extended with a concept of a *service task*. A *service task* is described as a regular computing task, and is able to start and terminate service processes, and runs throughout the workflow makespan.

However, running Chimbuko to extract performance data and its provenance with EnTK workflows is a non-trivial challenge. Initially, Chimbuko operates as an independent process, and it is necessary to launch and set it up correctly before any main component of the workflow can start. The original solution using EnTK involved creating an additional pipeline solely dedicated to handle Chimbuko processes, while other tasks were placed in separate pipelines that were pending for the completion of the Chimbuko's setup. This approach introduced significant complexity to the script and could potentially result in deadlocks due to the logic of launching different pipelines in EnTK. This issue was addressed with the introduced concept of a *service task* that allows launching the Chimbuko process automatically, before any other task. Having *service tasks* allows users to define when this process completed its setup correctly and is ready to analyze other tasks. It also facilitates the automatic termination of the Chimbuko process using a user-defined termination script, eliminating the need for additional checks to ensure the completion of other tasks. Improving communication protocols between services and EnTK is a currently ongoing task.

EnTK provides the execution environment with isolation capability. It allows a separation of the environment for computing tasks from the environment with the RCT stack, where the EnTK application is launched. Thus, using the Tuning Analysis Utilities (TAU) [26] in our applications (as Chimbuko does) requires to have it installed in a separate virtual environment to avoid potential version conflicts with packages used for the RCT stack. It is possible to have everything installed within a single environment, but having a separate execution environment gives more flexibility.

C. Darshan + RCT

Collecting workflow I/O performance data and metadata requires enabling Darshan within a particular EnTK workflow. While enabling I/O characterization, Darshan extends a set of libraries that should be used by computing tasks, i.e., it requires `LD_PRELOAD` environment variable to point to the `libdarshan.so`. We have tested this possibility to use Darshan while running a probe EnTK workflow on DOE HPC platforms - Polaris (ALCF) and Summit (OLCF). In our test experiments, we used both pre-installed (using module system `lmod`) and manually installed Darshan, and corresponding settings to load Darshan modules were provided within every task description.

Darshan has been used in conjunction with the LDMS [27] monitoring service in the past, providing an example of how to connect Darshan with monitoring infrastructure for runtime data access. We built on this example in accumulate the I/O performance metadata.

To test this integration, we focus on an inverse problem that is part of ExaLearn, an ECP (Exascale Computing Project) effort providing scalable AI/ML tools that enhance exascale-ready HPC applications through four pillars: surrogate models, inverse solvers, control policies, and design strategies [28]. Specifically, this problem tries to predict the structural parameters of crystalline solids directly from their Bragg profiles obtained by neutron scattering patterns. There are two main components in this application: simulation and training. Simulation only uses CPU resources and generates input data for the training, while training mainly uses GPU resources. This is a common paradigm for heterogeneous workflows at Exascale.

Specifically, we developed simplified mini-apps that target three execution patterns present in the inverse problem: (1) serial baseline that runs the entire simulation first to generate all training data, then runs the training; (2) serial sequence of simulation-training subtasks, which is equivalent to the serial baseline but reduces the resource utilization; (3) overlapped sequence of simulation-training (i.e., n^{th} training task runs concurrently with $(n+1)^{\text{th}}$ simulation task. These workflow mini-app patterns are designed to enable performance reproducibility of the original workflow without their domain-specific application dependencies. We tested the workflow mini-apps for each execution pattern on different LCF platforms, including Summit, Theta, and Polaris and validated the Darshan and RCT integration.

D. Mochi

The Mochi framework is designed for the rapid development of services for use in HPC platforms [24], and was used to develop the aforementioned Chimbuko service. In addition to its role in Chimbuko, Mochi is being used to implement data aggregation for the RECUP project, allowing for runtime analysis of workflow progress and comparison with prior executions. The current design calls for plug-ins to gather workflow behavior and performance data from RCT, I/O performance data from Darshan, and task performance data from Chimbuko. These data will be emitted from plug-ins and captured by a Mochi service using a publish-subscribe model similar to Kafka [29]. This event stream forms a temporal view of the progress of the workflow that can be directly interrogated for runtime analysis. The data will also be pushed into DataStates for persistent storage and further post-hoc analysis.

E. DataStates: Metadata organization and query optimization

Although our *Mochi* service will enable the scalable acquisition and unification of heterogeneous performance metadata, the access model it exposes (publish-subscribe) emphasizes a chronological flow of events in a set of streams. This access model is insufficient to study the performance reproducibility of different runs of the same HPC workflow, because of the complexity of the operations involved: we need to extract and/or calculate statistical properties of interesting events, identify what components of the workflow exhibit differences in performance at what point during the runtime based on the events, establish correlations between different types of events to identify the root cause of potential differences, etc. Implemented in a naive fashion, such operations would involve

multiple full passes over the performance metadata even when only a subset is needed, which is often a performance and scalability bottleneck. Thus, there is a need to reorganize and index the performance metadata in order to enable optimized queries specifically designed for performance reproducibility analytics.

To this end, we envision leveraging *DataStates* [25], a data management framework specifically designed to capture the evolution of intermediate data into a searchable lineage. Much like revision control systems, the lineage can be forked in different directions, which presents an opportunity to store the performance metadata of different workflow runs obtained from Mochi only once if it remains unchanged during runtime. Of particular interest are the foundational building blocks of DataStates, such as scalable multi-versioning ordered key-value stores used to persist the lineage in a compact fashion and to enable efficient key lookup queries and key iteration in sorted order [30]. Such building blocks are a promising starting point in the design and development of novel techniques to index and enable optimized queries for performance reproducibility analytics.

As a starting point, we analyzed the case of anomaly detection using *Chimbuko*, as applied to molecular dynamics simulations based on NWChem [31]. In this case, the performance metadata is collected into structured streams of events based on UnQLite, a binary format that supports nesting just like JSON, but more space efficient since it does not aim to be human-readable. The performance reproducibility analytics (detailed in Section III-G) needs to perform several time-consuming steps: (1) convert the performance metadata from the UnQLite format into the JSON format, which is understood by the high-level Python libraries used by the analytics; (2) reorganize the JSON content as Python Panda DataFrames; (3) repeatedly run analytics queries on top of the DataFrames. With respect to (1) and (2), we found that the conversion path from UnQLite to JSON to Panda DataFrames is not only a performance and scalability bottleneck, but it also leads to an explosion of memory utilization, making it unfeasible for large runs. With respect to (3), we identified several types of queries that are frequent and incur significant overheads: range scan, ordered entry search, join, and aggregation. Panda DataFrames lack efficient indexing and are not optimized for these types of queries, introducing further performance and scalability bottlenecks.

Based on these findings, we will explore techniques based on DataStates that emphasize high-throughput ingestion of the performance metadata from the event streams directly in binary format, for which we will design and implement, without additional conversions, scalable indexing and query support for the identified query types using high-level abstractions made available for Python. Thus, our proposed solution will act as an interoperability bridge between performance metadata collection and analytics, which improves performance and scalability, while reducing memory utilization.

F. Metadata capture and wrangling

To best support interoperability in our approach and generalizability with future systems we follow the RO-Crate metadata schema [32], and provide translational adjustments. RO-Crate is a lightweight set of guidelines to package digital Research Objects, e.g. datasets and software, with their metadata, using Zip containers and JSON-LD descriptors. Community-driven RO-Crate profiles, such as Workflow specifications and Workflow Provenance descriptions are agreed-upon schemas designed by relevant communities for specific purposes, thus fulfilling FAIR R1.3 principle. Our adjustments to a Workflow RO-Crate profile enable it to support data-focused workflows, optimized for ease of orchestration, and task-oriented ones, optimized for efficiency, thus bridging an important conceptual gap in the FAIR-ification of hybrid workflows. Our effort also supports interoperability with existing efforts that leverage RO-Crate, such as WorkflowHub[33], a workflow registry facilitating the discoverability and reusability of workflows.

The RO-Crate Workflow profile focuses on describing workflow steps, including software tools and the flow of input and output files between steps. A `Workflow` RO-Crate requires a `mainEntity` representing the source code of the workflow and must describe all of its components using the `hasPart` attribute. Components include output and reference files, described using entries with the `@type: File` attribute. The Provenance Run profile extends this specification by adding detailed step-by-step descriptions of the `mainEntity` with `howToStep` annotations to associate entities to an order of execution. The Provenance Run profile describes the inputs and outputs of each step in greater detail, using `input` and `output` attributes of each `howToStep`.

We found an inherent discrepancy between the expected format of an RO-Crate Workflow and one defined in RCT. In RCT, every component provides its level of abstraction and workflow description using EnTK following the Pipeline-Stage-Task model. *Task* is an abstraction of a computational task that contains information regarding an executable, its software environment, and its data dependencies. *Stage* is a set of tasks without mutual dependencies that can be executed concurrently. *Pipeline* is a list of stages where any following stage can be executed only after the execution of the previous one completes. This model conflicts with many workflow description languages that take a data flow-focus, such as the Common Workflow Language (CWL) [34] where a workflow is defined as a series of interdependent processes with flexible run order requirements and data flows, and used with RO-Crate.

To align the two paradigms, we decided upon the following equivalencies: first, an RCT *Pipeline* will use the RO-Crate `Workflow` definition. Second, RCT *Stages*, since they must be processed in linear order, will be considered sub-`Workflows` of the parent *Pipeline* Workflow by being listed under the *Pipeline*'s `hasPart` attribute. Third, *Tasks*, which may be completed in any order, will be defined as `howToSteps` without specified order.

Another important discrepancy between RCT and the CWL paradigms is their awareness of input and output files. The Provenance Run RO-Crate requires an exact definition of input and output files. This aligns with CWL, where a user defining a workflow must declare their input and output files such that the pipeline system can automatically determine a dependency graph of processes. RCT's EnTK system, however, is intentionally input-agnostic to dynamically run on HPC systems by allowing for implied or temporary dependencies. Therefore, to generate Provenance Run profiles EnTK must gain some awareness of input files: we are adding an annotation

method defining input and output files to EnTK's workflow definition accessible to users. We plan to use the Darshan system in this context to support automation and scalability of the annotations. Thus, EnTK will be able to generate a provenance graph describing the flow of files between *Stages*. This provenance graph will be used in generating an RO-Crate Provenance Run profile. It can also be compared with the file creation and deletion that RCT witnesses during runtime for reproducibility studies.

We have thus defined a format that will contain all metadata and workflow descriptions required to re-run an HPC workflow alongside metrics to validate performance reproducibility. However, one issue remains to be solved. Current RO-Crate examples imply that all relevant metadata is collected upon workflow completion, which requires a final omniscient component that polls all previous applications for their metadata. Such a component is inconvenient in high performance and parallelized systems; the creation of a workflow's profile could become desynchronized from the workflow's completion, resulting in workflows failing to be annotated. Metadata may also become inaccessible upon workflow completion, as earlier tools may remove temporary run information upon workflow completion for performance reasons. Furthermore, as additional software tools are added to the pipeline or depreciated ones are removed, this omniscient component must be adjusted to reflect these changes.

We are investigating a more granular metadata collection system than shown in figure 1. Rather than collecting a complete RO-Crate profile of metadata upon workflow collection, we will design the concept of partial profiles for each component. Each system will be responsible for a miniature profile describing the metadata generated during its process. For example, EnTK controls the pipeline execution and therefore EnTK will be responsible for generating a partial RO-Crate profile describing the pipeline, its stages and tasks, and the hardware these tasks were run on. On the other hand, application-specific metadata will be extracted separately, as illustrated in Section III-G for the Chimbuko use case. Such a separation of metadata collection responsibilities provides better modularity and robustness to changes in the overall architecture compared with monolithic approaches, while facilitating better cross-tool interoperability.

G. Use Case: Performance reproducibility

Our preliminary work [35] shows that in a production HPC facility, using the default configuration option to run an application can cause up to 4x variation across the repeated runs of the same application. This observation signifies the need for performance reproducibility studies since such drastically different execution time of the same application means that jobs may get cancelled without finishing. Such repeated runs can prolong the turnaround time of science.

As a customer of the data generated by applications, the performance reproducibility framework requires configuration parameters annotated with their names (metadata) and the execution times for each instance of the run. Specifically, for the data collected by the Chimbuko framework, the entire collection of the anomalous instances are stored in an UnQLite-based provenance database. For offline analysis, we use a Python script to query the database and generate a JSON dump of the entire dataset, where each entry corresponds to an event. Each event contains information about the application, process, and thread id, node name where the anomaly appeared, timestamp, anomaly severity, a score reflecting how probable it is that a function execution is anomalous, hardware performance counters, callstack pertaining to the anomalous function, and execution time. From a user's perspective, the analysis pipeline expects the data to be annotated, the problem to be described, and the data to be accessible during analysis. In our preliminary work with the Panda's DataFrame abstraction for analyzing the characteristics of the performance anomaly instances, we encountered additional overhead due to multi-step conversion from database to JSON to Panda's DataFrame. This barrier can be lowered by the RECUP framework as it can provide efficient indexing and query support (as mentioned in Section III-E) that avoids the performance, scalability, and memory utilization bottlenecks we encountered due to repeated conversion and revisiting of the performance metadata.

IV. Conclusions and Future Work

In this paper, we proposed RECUP, a comprehensive performance reproducibility framework that integrates a diversity of tools in order to manage the entire life-cycle of metadata needed to enable the study of performance reproducibility. In particular, we highlighted the importance of performance metadata collection, curation, aggregation, streaming, indexing and query optimization, which is necessary in order to enable efficient analytics that involves multiple repeated passes over the performance metadata from multiple different perspectives. Furthermore, we discussed the challenges and preliminary work we undertook to make these tools interoperable. In doing so, we conclude with several important observations: (1) given the complexity of the life-cycle of the performance metadata, a large number of specialized tools is needed, each of which addresses specific steps and needs to interact with at least one other tool; (2) the interoperability between pairs of tools is non-trivial due to a mismatch between intermediate representations and their semantics; (3) naive solutions to convert intermediate representations are inefficient and lead to large overheads; and (4) designing tools with efficient metadata solutions and FAIR interoperability in mind from inception is necessary for performance reproducibility, especially as ever larger datasets pouring out of DOE experimental facilities will need to be processed on LCFs. Based on these observations, in future work we plan to finalize the implementation of the proposed architecture and the interoperability between the tools, which enables us to obtain an end-to-end solution that we will evaluate and compare with baseline approaches using comprehensive experiments.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under grant 0269227 and contract DEAC02-06CH11357.

References

[1] A. Al-Saadi, D. H. Ahn, Y. Babuji, K. Chard, J. Corbett, M. Hategan, S. Herbein, S. Jha, D. Laney, A. Merzky *et al.*, "Exaworks: Workflows for exascale," in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2021, pp. 50–57.

[2] P. V. Coveney, D. Groen, and A. G. Hoekstra, "Reliability and reproducibility in computational science: Implementing validation, verification and uncertainty quantification in silico," p. 20200409, 2021.

[3] B. Hu, S. Canon, E. A. Eloe-Fadrosh, M. Babinski, Y. Corilo, K. Davenport, W. D. Duncan, K. Fagnan, M. Flynn, B. Foster *et al.*, "Challenges in bioinformatics workflows for processing microbiome omics data at scale," *Frontiers in Bioinformatics*, vol. 1, p. 826370, 2022.

[4] P. Vaillancourt, B. Wineholt, B. Barker, P. Deliyannis, J. Zheng, A. Suresh, A. Brazier, R. Knepper, and R. Wolski, "Reproducible and portable workflows for scientific computing and hpc in the cloud," in *Practice and Experience in Advanced Research Computing*, 2020, pp. 311–320.

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

[6] O. E. Gundersen and S. Kjensmo, "State of the art: Reproducibility in artificial intelligence," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[7] O. E. Gundersen, K. Coakley, C. Kirkpatrick, and Y. Gil, "Sources of irreproducibility in machine learning: A review," *arXiv preprint arXiv:2204.07610*, 2022.

[8] B. A. Plale, T. Malik, and L. C. Pouchard, "Reproducibility practice in high-performance computing: Community survey results," *Computing in Science & Engineering*, vol. 23, no. 5, pp. 55–60, 2021.

[9] L. Pouchard, T. Islam, and B. Nicolae, "Challenges for implementing fair digital objects with high performance workflows," *Research Ideas and Outcomes*, vol. 8, p. e94835, 2022.

[10] S. Stall, L. Yarmey, J. Cutcher-Gershenfeld, B. Hanson, K. Lehnert, B. Nosek, M. Parsons, E. Robinson, and L. Wyborn, "Make scientific data fair," *Nature*, vol. 570, no. 7759, pp. 27–29, 2019.

[11] D. Dai, R. B. Ross, P. Carns, D. Kimpe, and Y. Chen, "Using property graphs for rich metadata management in hpc systems," in *2014 9th parallel data storage workshop*. IEEE, 2014, pp. 7–12.

[12] H. Tang, S. Byna, B. Dong, J. Liu, and Q. Koziol, "Someta: Scalable object-centric metadata management for high performance computing," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 359–369.

[13] J. M. Wozniak, Z. Liu, R. Vescovi, R. Chard, B. Nicolae, and I. Foster, "Braid-db: Toward ai-driven science with machine learning provenance," in *Smoky Mountains Computational Sciences and Engineering Conference*. Springer, 2021, pp. 247–261.

[14] C. Goble, S. Cohen-Boulakia, S. Soiland-Reyes, D. Garijo, Y. Gil, M. R. Crusoe, K. Peters, and D. Schober, "Fair computational workflows," *Data Intelligence*, vol. 2, no. 1-2, pp. 108–121, 2020.

[15] R. F. Da Silva, H. Casanova, K. Chard, I. Altintas, R. M. Badia, B. Balis, T. Coleman, F. Coppens, F. Di Natale, B. Enders *et al.*, "A community roadmap for scientific workflows research and development," in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2021, pp. 81–90.

[16] A. Jokanovic, J. C. Sancho, G. Rodriguez, A. Lucero, C. Minkenberg, and J. Labarta, "Quiet neighborhoods: Key to protect job performance predictability," in *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2015, pp. 449–459.

[17] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir, "Scheduling the i/o of hpc applications under congestion," in *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2015, pp. 1013–1022.

[18] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda *et al.*, "Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 2015, pp. 1–12.

[19] Y. Fan, P. Rich, W. E. Allcock, M. E. Papka, and Z. Lan, "Trade-off between prediction accuracy and underestimation rate in job runtime estimates," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 530–540.

[20] S. Ramesh, M. Titov, M. Turilli, S. Jha, and A. Malony, "The ghost of performance reproducibility past," in *2022 IEEE 18th International Conference on e-Science (e-Science)*. IEEE, 2022, pp. 513–518.

[21] M. Turilli, V. Balasubramanian, A. Merzky, I. Paraskevakos, and S. Jha, "Middleware building blocks for workflow systems," *Computing in Science & Engineering*, vol. 21, no. 4, pp. 62–75, 2019.

[22] P. Carns, "Darshan," in *High performance parallel I/O*. Chapman and Hall/CRC, 2014, pp. 351–358.

[23] C. Kelly, S. Ha, K. Huck, H. Van Dam, L. Pouchard, G. Matyasfalvi, L. Tang, N. D'Imperio, W. Xu, S. Yoo *et al.*, "Chimbuko: A workflow-level scalable performance trace analysis tool," in *ISAV'20 In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, 2020, pp. 15–19.

[24] R. B. Ross, G. Amvrosiadis, P. Carns, C. D. Cranor, M. Dorier, K. Harms, G. Ganger, G. Gibson, S. K. Gutierrez, R. Latham *et al.*, "Mochi: Composing data services for high-performance computing environments," *Journal of Computer Science and Technology*, vol. 35, no. 1, pp. 121–144, 2020.

[25] B. Nicolae, "DataStates: Towards Lightweight Data Models for Deep Learning," in *SMC'20: The 2020 Smoky Mountains Computational Sciences and Engineering Conference*, Nashville, United States, 2020, pp. 117–129. [Online]. Available: https://hal.inria.fr/hal-02941295

[26] S. S. Shende and A. D. Malony, "The tau parallel performance system," *The International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.

[27] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden *et al.*, "The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 154–165.

[28] F. J. Alexander, J. Ang, J. A. Bilbrey, J. Balewski, T. Casey, R. Chard, J. Choi, S. Choudhury, B. Debusschere, A. M. DeGennaro *et al.*, "Co-design center for exascale machine learning technologies (exalearn)," *The International Journal of High Performance Computing Applications*, vol. 35, no. 6, pp. 598–616, 2021.

[29] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, 2011, pp. 1–7.

[30] B. Nicolae, "Scalable Multi-Versioning Ordered Key-Value Stores with Persistent Memory Support," in *IPDPS 2022: The 36th IEEE International Parallel and Distributed Processing Symposium*, Lyon, France, 2022, pp. 93–103. [Online]. Available: https://hal.inria.fr/hal-03598396

[31] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J. J. Van Dam, D. Wang, J. Nieplocha, E. Aprà, T. L. Windus *et al.*, "Nwchem: A comprehensive and scalable open-source solution for large scale molecular simulations," *Computer Physics Communications*, vol. 181, no. 9, pp. 1477–1489, 2010.

[32] S. Soiland-Reyes, P. Sefton, M. Crosas, L. J. Castro, F. Coppens, J. M. Fernández, D. Garijo, B. Grüning, M. La Rosa, S. Leo *et al.*, "Packaging research artefacts with ro-crate," *Data Science*, vol. 5, no. 2, pp. 97–138, 2022.

[33] C. Goble, S. Soiland-Reyes, F. Bacall, S. Owen, A. Williams, I. Eguinoa, B. Droesbeke, S. Leo, L. Pireddu, L. Rodríguez-Navas, J. M. Fernández, S. Capella-Gutierrez, H. Ménager, B. Grüning, B. Serrano-Solano, P. Ewels, and F. Coppens, "Implementing FAIR Digital Objects in the EOSC-Life Workflow Collaboratory," Mar. 2021, white paper. [Online]. Available: https://doi.org/10.5281/zenodo.4605654

[34] M. R. Crusoe, S. Abeln, A. Iosup, P. Amstutz, J. Chilton, N. Tijanić, H. Ménager, S. Soiland-Reyes, B. Gavrilović, C. Goble, and T. C. Community, "Methods included: Standardizing computational reuse and portability with the common workflow language," *Commun. ACM*, vol. 65, no. 6, p. 54–63, may 2022. [Online]. Available: https://doi.org/10.1145/3486897

[35] T. Patki, J. J. Thiagarajan, A. Ayala, and T. Z. Islam, "Performance optimality or reproducibility: That is the question," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–30.