Novel Proposals for FAIR, Automated, Recommendable, and Robust Workflows

Ishan Abhinit*, Emily K. Adams*, Khairul Alam†, Brian Chase*, Ewa Deelman[xi], Lev Gorenstein‡
Stephen Hudson§, Tanzima Islam¶, Jeffrey Larson§ Geoffrey Lentner‡, Anirban Mandal‖, John-Luke Navarro§
Bogdan Nicolae§, Line Pouchard**, Rob Ross§ Banani Roy†, Mats Rynge[xi], Alexander Serebrenik††
Karan Vahi[xi], Stefan Wild§, Yufeng Xin‖, Rafael Ferreira da Silva‡‡, Rosa Filgueira[x]
*Indiana University, Bloomington, IN, USA †University of Saskatchewan, Saskatoon, SK, Canada
‡Purdue University, West Lafayette, IN, USA §Argonne National Laboratory, Lemont, IL, USA
¶Texas State University, San Marcos, TX, USA ‖Renaissance Computing Institute, Chapel Hill, NC, USA
**Brookhaven National Laboratory, Upton, NY, USA ††Eindhoven University of Technology, Eindhoven, Netherlands
[xi]University of Southern California, Marina del Rey, CA, USA ‡‡Oak Ridge National Laboratory, Oak Ridge, TN, USA
[x]University of St Andrews, St Andrews, UK

***Abstract*—Lightning talks of the Workflows in Support of Large-Scale Science (WORKS) workshop are a venue where the workflow community (researchers, developers, and users) can discuss work in progress, emerging technologies and frameworks, and training and education materials. This paper summarizes the WORKS 2022 lightning talks, which cover five broad topics: data integrity of scientific workflows; a machine learning-based recommendation system; a Python toolkit for running dynamic ensembles of simulations; a cross-platform, high-performance computing utility for processing shell commands; and a meta(data) framework for reproducing hybrid workflows.**

***Index Terms*—scientific workflows, FAIR, high performance computing, data integrity, ensembles, machine learning.**

I. Introduction

Scientific workflows have been almost universally used across scientific domains and have underpinned some of the most significant discoveries of the past several decades. As workflows have been adopted by a number of scientific communities, they are becoming more complex and require more sophisticated workflow management capabilities [1]. In this context, the workshop on Workflows in Support of Large-Scale Science (WORKS) has positioned itself as the primary venue for workflow researchers and developers to share and discuss innovative ideas to enhance the current workflow research and development landscape. Specifically, WORKS' lightning talks provide a venue where members of the community can introduce short talks on works in progress, emerging technologies and frameworks, and training and education materials to lower the entry barrier and thus increase adoption. This paper provides overviews of the five lightning talks from the 17th edition of the workshop (WORKS 2022):

Data integrity of scientific workflows (Section II) – This work describes two popular cybersecurity classification frameworks – OSCRP and MITRE ATT&CK®; can be leveraged to systematically model threats to the integrity of scientific workflows and data in a research setting. We enumerate non-malicious and malicious threats to the integrity of scientific workflows, and present the relevant assets, concerns, avenues of attacks and impact of the threats in typical scientific workflow execution scenarios.

Recommendation System (Section III) – This work proposes a recommendation system to recommend tools/sub-workflow using machine learning approaches to help scientists create optimal, error-free, and efficient workflows by analyzing existing workflows in various workflow repositories.

libEnsemble (Section IV) – A Python toolkit for running dynamic ensembles of simulations. libEnsemble aims to minimize the effort of the user in describing their workflow via generator and simulator functions written in Python, and to maximize code reuse by maintaining a library of existing functions. Example generator functions perform optimizations, train models, and test candidate solutions. This work highlights how libEnsemble's dynamic features have enabled practical multi-fidelity workflows.

HyperShell v2 (Section V) – An elegant, cross-platform, high-performance computing utility for processing shell commands over a distributed, asynchronous queue. It is a highly scalable workflow automation tool for *many-task* scenarios. HyperShell was originally created several years ago at Purdue University to meet the specific unmet needs of researchers not satisfied by existing solutions. Here we will outline the context for its existence and focus on some unique capabilities it offers.

A (meta)data framework for reproducing hybrid workflows

(Section VI) – A conceptual framework and methods to extract and share data and metadata necessary for reproducibility in the context of complex hybrid workflows executed at extreme scale. The framework targets Digital Objects required to reproduce results and performance: it captures, fuses, and analyzes (meta)data to select parameters influencing reproducibility, and make them FAIR Digital Objects for re-use.

II. Modeling Data Integrity Threats for Scientific Workflows Using OSCRP and MITRE ATT&CK®

By: Ishan Abhinit, Emily K. Adams, Brian Chase, Anirban Mandal, Yufeng Xin, Karan Vahi, Mats Rynge, and Ewa Deelman

With the rise in scale and complexity of scientific workflows, it is extremely important to assure the the integrity of the scientific data, as they are processed and transmitted on distributed infrastructures. When data integrity is not preserved, computational workflows can fail and result in increased computational cost due to reruns, or worse, results can be corrupted compromising the scientific outcomes [2]. Recent works, e.g. [3] are initial efforts to ensure that science workflows and data transfers are guarded against data integrity errors that might arise in complex distributed systems. However, these works have not addressed the diagnosis and pinpointing of the root cause of data integrity errors. Any such analysis will need to incorporate the threat models for data corruption, both non-malicious and malicious threats. In this work, we leverage two existing popular cybersecurity frameworks, Open Science Cyber Risk Profile (OSCRP) [4] and MITRE ATT&CK knowledge base [5] to build threat models for scientific workflows to pinpoint the root cause of unintentional (*i.e.* non-malicious) integrity errors [6], and uncover adversarial tactics and techniques (*i.e.* malicious) [7] that threaten data integrity within scientific workflows.

A. Non-malicious Data Integrity Threat Model Using OSCRP

Open Science Cyber Risk Profile (OSCRP) is designed to assess cybersecurity risks related to open science projects.Open science research often has unique cybersecurity concerns and OSCRP provides a catalog of typical scientific research assets and the associated risks to research activities. Steps involved in the OSCRP risk profiling process are: 1) Identify the stakeholders; 2) Create an asset inventory; 3) Examine the concerns, consequences and avenues of attack for each mission critical science asset; and 4) for each relevant concern, identify the vectors/methods of attack that could cause the concern to be realized.

1) Categorizing Scientific Workflow Assets using OSCRP: We first translated the OSCRP asset classifications into vernacular suited explicitly for scientific workflows. The resultant asset classification, shown in Figure 1 with their associated OSCRP asset class (in parenthesis) are: I. Transient workflow (”Internal” data); II. Data products (”Public” data); III. Metadata (”Accounting” data), IV. Researcher system (”Desktop”); V. Workflow management system (”Workflow”); VI. Computational systems (”Servers”); VII. Data storage systems (”File storage”); VIII. Network systems (”Networks”). By subsequently overlaying the scientific workflow as enacted by a representative workflow management system, Pegasus [8], with our OSCRP asset classification, indicated with red markers in Figure 1, it enabled us to enumerate possible sources of data integrity errors. Thus, we were able to derive non-malicious threats and conduct an impact analysis to the integrity of the scientific workflow.

TABLE I
Two Excerpts from ”Detailed Analysis of Assets to Concerns and Avenues of Attack” Table (Adapted)

OSCRP Asset	Concern(s)	Consequence/Impact	Integrity Degradation
I	Corrupted data, incorrect data, or lost data	Workflow producing incorrect/invalid results	Issues with data processing, issue with sensor equipment
V	Lost or incorrect process	Workflow producing incorrect/invalid results	Issues with storage

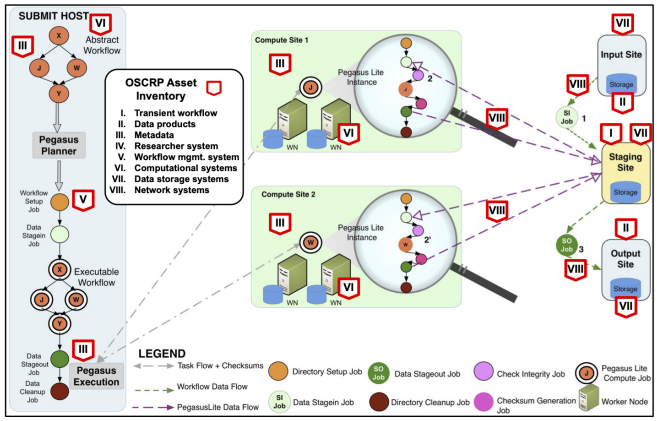

Fig. 1. OSCRP Asset Mapping diagram for Workflows (Adapted).

2) Findings from Non-Malicious Data Integrity Threat Model: We started our non-malicious scientific workflow threat modeling by looking at each asset to determine how and where the data integrity can be degraded. We then identified concerns (*i.e.* a negative change to an asset that impacts a research activity) for these asset. Finally, we determined the vectors or methods of integrity degradation to complete our non-malicious integrity analysis. Table I exhibits two examples from our first threat model itemizing potential non-malicious impacts to the integrity of scientific workflows.

Malicious Threat Model Using MITRE ATT&CK – The MITRE ATT&CK Enterprise knowledge base is used as a foundation for the development of targeted cybersecurity threat models and methodologies. We used the MITRE ATT&CK framework of ATT&CK Tactics and Techniques to conduct a scoped impact analysis enumerating malicious attacks against data integrity within scientific workflows.

TABLE II
TWO EXCERPTS FROM ATT&CK TACTIC IMPACT ANALYSIS (ADAPTED)

ATT&CK Tactic & Technique	**OSCRP Asset**	**Attack Type**	**Data Integrity Concern** (An adversary may...)	**Scientific Workflow Impact**
[Impact Tactic] Data Destruction T1485	I II III	Direct	Destroy data and files. Render stored data irrecoverable by forensic techniques.	Scientific workflow cannot be initiated or executed. Data lost.
[Execution Tactic] Inter-Process Communication T1559	I II III	Indirect	Abuse inter-process communication (IPC) mechanisms for local code or command execution	Scientific workflow cannot be initiated, executed, is degraded, is compromised and/or producing invalid results.

Defining Data Assets using OSCRP and Attack Type – Prior to leveraging ATT&CK adversarial Tactics and Techniques to develop a threat model for malicious attacks, we scoped our focus to only address the integrity of data within the workflows. Thus, the following subset of OSCRP asset classifications specifically involving data were carried over from the non-malicious threat model for this analysis: I. Transient Data; II. Data Products; and III. Metadata.

We further scoped our malicious threat model by defining attack types that explicitly impact scientific data integrity within scientific workflows: Direct Attacks, defined as malicious activity directly targeting research data with the goal to impact data integrity (*e.g.* delete, alter); Indirect Attacks, defined as malicious activity targeting an asset or process that interfaces with data. *(e.g*. altering scientific software), but did not include General Attacks, defined as malicious activity that impacts the security of assets, functions, and personnel supporting research activities.

Findings from Malicious Data Integrity Threat Model – The analysis culminated by determining which MITRE ATT&CK Tactics and Techniques were relevant to malicious attacks against scientific data integrity per OSCRP-derived asset class and attack type. Of the thirteen MITRE ATT&CK Tactics, two ATT&CK Tactics, with a subset of seven ATT&CK Techniques each, were identified to precipitate malicious activity, which *directly* or *indirectly* impacts the integrity of scientific data.

- Impact [TA0040] - The adversary is trying to manipulate, interrupt, or destroy systems and data.
- Execution [TA0002] - The adversary is trying to run malicious code.

Table II contains two examples of ATT&CK Techniques that we determined directly or indirectly impacted data integrity (*i.e.* Data Integrity Concern, and Scientific Workflow Impact).

B. Application & Future Work

By systematically identifying tactics and techniques of malicious threats to data within scientific workflows, data stewards will be equipped to proactively design scientific workflows to ensure data integrity, better understand how malicious attacks against data integrity are executed, and proactively build detections and defenses to protect scientific data.

In the future, we will leverage the analysis derived from our two threat models for pinpointing root causes of workflow data integrity errors, by informing Machine Learning (ML) based analysis models to more closely align to threats observed in real systems and workflow execution scenarios. While one can inject errors based on anecdotal evidence during ML model development, injecting errors guided by malicious and non-malicious threat models should be a far superior technique, grounded in well-established cybersecurity frameworks.

III. RECOMMENDING TOOLS AND SUB-WORKFLOWS FOR SCIENTIFIC WORKFLOW MANAGEMENT SYSTEMS

By: Khairul Alam, Banani Roy, and Alexander Serebrenik

The objective of recommender systems is to help people find suitable, exciting, and newly released products. The task of recommender systems is to turn data on users and their preferences into predictions of users' possible future likes and interests [9]. Recommender systems have been used in online shopping, travel booking, and media service providers for many years. It is also used extensively in the scientific community for scientific literature searches to help scientists explore relevant and recent papers quickly. The recommended products are primarily chosen based on past usage and purchasing patterns. Decreasing the data storage and processing cost led to the significant improvement of recommender systems by making predictions on the available data. Commercial companies like Alibaba and Amazon use their customers' preferences to select products from an extensive collection. Production companies like YouTube and Netflix suggest new songs, movies, etc., based on the previous usage of the customer. In short, recommender systems make life smoother for the users to find appropriate things among the ocean of products. Recommender systems differ in how they analyze data sources to develop notions of affinity between users and items, which can be used to identify well-matched pairs [10]. The successful usage of recommendation systems by companies to find relevant things encouraged us to create a tools/sub-workflows recommendation system in SWfMSs

Workflows are becoming essential in analyzing scientific data, and there are hundreds of scientific workflow management systems where researchers can create, collaborate and share workflows for their analysis. However, a critical issue for a workflow is that it is not always state-of-the-art or even valid at all. A workflow may run on a system without creating an issue and produce some results, but the generated result may not always be correct due to the wrong selection of tools and techniques. There are several [11]–[16] publicly available workflow repositories, and there are also several tools/workflow recommendation systems [17]–[19] available which usually recommends based on the previous usages of the tools. But tools and techniques are becoming updated every now and then. Also, many new optimized tools are being added to the workflow management system, many tools are

becoming deprecated, and many workflows have errors. So recommendation system based on these workflows will surely recommend the wrong tools/sub-workflows at some point in the workflow construction. So in this paper, we proposed a tools/sub-workflows recommendation system. While developing the system, we will consider the usage frequencies of tools, high-quality and low-quality sequences, usage period, workflow naming, tagging, annotations, and other parameters.

SWfMSs contain thousands of tools to analyze scientific datasets, and they are increasing rapidly. We also noticed that scientists are now more eager to publish their workflows and datasets so that other users can benefit from them. Galaxy SWfMS has several workflow histories repositories [20]–[22], and is the most popular SWfMS to date. So for conducting this research, we explored Galaxy Main Server [20] published histories and identified that a significant percentage of workflows have issues with them, and we are expecting similar results for the other SWfMSs. So recommendation systems based on the existing workflows may provide undesired suggestions. For our case, in the beginning, we will identify the poorly designed workflow using manual (inconsistent but running workflows) and automated (tools compatibility, obsolete tools, broken workflow, and so on) processes and discard them so that our system can suggest the best combinations of tools while constructing a workflow.

At first, we will explore all available Galaxy workflow repositories to gather the reusable workflows and discard error-containing, irrelevant and obsolete workflows. After that, using machine learning approaches, we will build an intuitive and user-friendly tools/sub-workflow recommendation system using reusable workflows. We also have a plan to work with other SWfMSs. We believe that our work will contribute in the following ways:

1) Our analysis data will help raise awareness to create workflow appropriately so that researchers can understand and re-use it without facing too many difficulties.
2) Our proposed system will save time researchers waste in creating erroneous or less optimal workflows by choosing improper tools which produce unexpected results.
3) It will relieve researchers from memorizing a vast amount of tools and also increase the accessibility.
4) It will promote high-quality tools by checking the previous (a certain period) usage frequencies and downgrading those having lower usage frequencies.

A. Proposed Recommendation System

Our proposed system will suggest tools/sub-workflows for designing a new workflow by deriving structural information from the existing shared reusable workflows from the workflow repository. At first, we will discard all irrelevant, error-containing workflows. After that, based on the remaining workflows, our system will predict tools/sub-workflows for completing the new workflow. We will consider each workflow as a graph to easily find out which node is connected with which node and store them in a database. The co-relation between nodes will be used to rank the tools. The higher ranked nodes will get high preferences to be selected. Besides node frequency, we will also collect additional information like pipeline structure, sequential pattern, tools usage pattern, annotations of the tools, tools compatibility, annotation of tools and workflows, tagging information, and proper workflow naming. We will assign weights to these nodes since they might help workflow designers better understand a workflow. If any new tool is developed, the system will be able to recommend the tool wherever appropriate. Sometimes, tools with low usage frequencies are useful, like high usage tools; these criteria will also be maintained while suggesting tools.

B. Related Work

Researchers developed several recommendation systems to simplify workflow construction and scientific analysis. Mass spectrometry-based proteomics [23] used EDAM and semantic annotations of tools to compose a workflow automatically. DiBernardo et al. [24] used data types to create a workflow automatically. Koop et al. built *VisComplete*, a system to aid users in creating visualization pipelines based on VisTrails SWfMS using a database of previously created visualization pipelines but did not consider the correctness of the previous pipelines. All these approaches apply to a limited set of bioinformatics analyses and are subject to error as the analysis tools become updated frequently. Kumar et al. [17] developed a model for suggesting tools while constructing a workflow using a deep learning approach by analyzing workflows available in the European Galaxy server. But they did not consider the type compatibility, tools annotations, errors, obsolete tools, and so on issues while suggesting the tools. To the best of our knowledge, our proposed approach is the first to consider previous workflow correctness while suggesting tools/sub-workflow while constructing a new workflow.

C. Conclusion and Future Work

For proposing this research work, we explored various workflow repositories, extracted workflows, and identified several issues in the existing workflows. We also explored state-of-the-art tools and techniques for recommending workflows in scientific workflows and found that none of them considered existing workflows' correctness in building their systems. So, we planned to work on it. As designing scientific workflow is not an easy task but vital for many scientific domains, our future plan is to build a recommender system to suggest tools while creating workflows based on error-free reusable workflows.

IV. libEnsemble: Flexible Workflows through Dynamic Assignment of Workers and Resources

By: Stephen Hudson, Jeffrey Larson, John-Luke Navarro, and Stefan M. Wild

libEnsemble [25], [26] coordinates computations via a simple *manager-workers* paradigm, which can run on one of three communication mediums: MPI (via `mpi4py`), multiprocessing (via Python's built-in module), and TCP (for distributed/cloud-based environments).

Workers call the user-provided Python *generator* and *simulator* functions to perform any type of computation. An allocation function on the manager, also exposed to users, controls at any time which workers are running generators/simulators, and what data is given to them. The full ensemble data is always maintained in a central data structure known as the history array (a NumPy structured array).

Each generator and simulator function can be either persistent or non-persistent. A non-persistent function simply carries out a given task in a fire-and-forget manner, while persistent functions continue to run on workers while communicating with the manager. This is useful, for example, to maintain the state of a model, or to report back intermediate results from a still-running simulation. Persistent workers can self-terminate based on a local condition or be shut down by the allocation function.

In addition to updating simulation input/output data, manager/worker communications may be used to request cancellation of previously issued points or to send kill signals to terminate running simulations.

A workflow can consist of a combination of persistent and non-persistent workers, with one or multiple generators, enabling multiple-stage workflows. This includes the reassignment of workers between running generators or simulators as the ensemble progresses.

libEnsemble also manages resources including both nodes and partitions within nodes. The detected resources are divided into a given number of resource sets and each simulation can be given some number of resource sets alongside other inputs.

The task scheduler aims to assign resources to each task in an efficient way and splits evenly across nodes if necessary. Scheduler behavior can be customized by user options, and by inheritance if required.

A user (simulator or generator) function can access all information about local resources in order to assign environment variables or run-line commands (such as requesting GPUs). This feature has been demonstrated on systems including Summit, Perlmutter, Spock, Crusher, and multiple clusters, including test systems using Intel GPUs. By handling resources in a consistent way, libEnsemble enables resource management to work seamlessly across these systems, despite their different application level scheduling capabilities.

A collaboration between DESY (Deutsches Elektronen-Synchrotron), Hamburg University, Lawrence Berkeley National Laboratory, and Argonne National Laboratory has used libEnsemble for tuning inputs to Laser-plasma accelerators using multi-fidelity simulations on GPUs. This work has involved various optimization libraries such as **`Dragonfly`** and **`BoTorch`** and simulation codes including **`Wake-T`**, **`FBPIC`** and the ECP application **`WarpX`**. This work has examples demonstrating an order of magnitude performance gain over the single-fidelity method [27].

Figure 2 shows an example of multi-fidelity versus single-fidelity simulations using Dragonfly and BoTorch optimization methods combined with FBPIC simulations. The methods observe simulation output and request subsequent simulations at various fidelity levels. libEnsemble dynamically allocates CPU/GPU resources as requested by the methods. Computational efficiency is increased as less-expensive, lower fidelity simulations can guide numerical optimization methods.

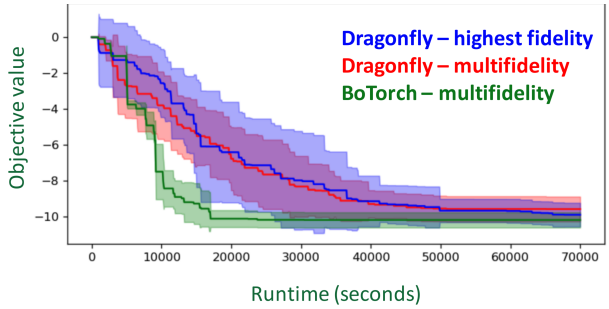

Fig. 2. Progress from ten replications of libEnsemble+FBPIC with two multifidelity methods and a single (highest) fidelity method. Objective value is computed from the highest fidelity simulation.

V. HyperShell v2: A Better Workflow Automation Tool for Many-Task Computing

By: Geoffrey Lentner and Lev Gorenstein

Many-task computing refers to workflows composed of some large number of relatively small tasks all of which are only loosely coupled if not entirely independent. Unlike traditional high-performance computing (HPC) workloads such as large-scale simulations, these many-task scenarios are both more flexible and yet potentially more complex. Such use-cases include data processing pipelines, machine-learning experiments, genomics or bioinformatics tasks, parameter sweeps in a calculation, or any such collection of tasks.

Researchers will often write such workflows using their chosen application language (Python, R, MATLAB, etc.) because the task itself is defined within such code already. There exist many frameworks or packages within most of these languages that allow for parallel and even distributed execution. Some of these are easier to work with than others. This can be difficult for researchers, fraught with inefficiencies, lacking flexibility, and unnecessarily time consuming at best. In all cases, we suggest the best practice is to instead define tasks as singular execution elements and abstract the scale-out procedure using some kind of workflow automation tool.

In a high-performance computing environment this is most readily accomplished with the available workload manager or scheduler; e.g., SLURM [28]. However, it is common with these sorts of many-task workloads to have many thousands of tasks or more. Submitting these alone may be an option, but even if the scheduler can manage the throughput, the site administrators may not allow such volumes of jobs. Additionally, queuing and startup overhead for a large number of small jobs could accumulate to prohibitively long total workflow execution time. Instead, the accepted paradigm is to submit a single *pilot job* to the scheduler that merely acquires

TABLE III
FEATURE COMPARISON

	Distributed	Restart / Failures	Scalable	Observable	Cross Platform	Persistent
HyperShell	X	X	X	X	X	X
xargs						
srun	X		X			
GNU Make		X				
GNU Parallel	X	X				
ParaFly		X				
Launcher	X	X	X			
TaskFarmer	X	X				

the resources and launches an external workflow automation tool to scale-out within the single allocation.

A. Solution Category

There already exists quite a lot of work in this space with some projects dominating within a particular scientific domain or focusing on a particular use-case. Some of these solutions are feature rich and delve into the related challenge of data locality by integrating data movement between elements of the workflow system. And often the necessary prerequisite is to define the workflow within some domain-specific language (DSL) and/or provide an actual directed acyclic graph (DAG) of tasks and their dependency relationships; such as *Swift* [29], *GNU Make* [30], or *Snakemake* [31]. Workflow systems can even be defined entirely within a library particular to a specific programming language, such as *Parsl* [32].

We are going to specifically confine ourselves within a narrowly defined category of solutions that exist with a particular focus on simplicity of design and use. These tools simply operate on a monolithic continuous stream a single command lines, typically allowing for these lines to be provided by file or from standard input.

Several solutions exist in this space; some of them are common to Linux and BSD systems; *xargs*, or *GNU Parallel* [33]. Some of these tools are born of other projects, such as *ParaFly* from Trinity [34]. Others are developed by HPC centers themselves, such as *Launcher* at TACC [35], *TaskFarmer* at NERSC [36], and now *HyperShell* at Purdue University [37].

B. Features

We recently presented our work on the development and release of *HyperShell* v2 at *PEARC22* [38]. There we outlined the overall goal of the project, feature comparisons with some of the alternatives listed here, and made the essential value proposition. See Table III.

HyperShell is easy to understand in simple cases yet feature rich in more complex scenarios. We see users go from serial and/or manual execution to robust and distributed in only a few minutes. The ergonomics are such that even novice researchers can use the tool effectively with little training.

When using the top-level *cluster* mode the interface is very close to that of *GNU Parallel* with robust template expansions but better scaling because of it's two-tiered client-server architecture and task bundling. So far, we have measured the responsiveness and throughput of *HyperShell* up to 1,000 nodes on Purdue's new Anvil system, an XSEDE/ACCESS resource [39].

While the common scenario is to target a simple input task file from the command line, *HyperShell* can actually run as a persistent server with a database in-the-loop (e.g., PostgreSQL) and act as a stand-alone scheduling system with individual task submissions. For millions of tasks it offers a rich query tool to search task history. *HyperShell* can also be embedded within a Python application as a library and used to drive such distributed task execution using asynchronous threads.

Possibly the most novel aspect of *HyperShell* is not just that it can run anywhere Python can run, but that its cross-platform nature transparently supports heterogeneous execution modes with the server and clients being on different platforms or operating systems. Indeed, one of the founding use-cases for the project was to allow for tens of thousands of 15-second tasks running within a Windows environment; such that we couldn't afford to throw away the virtual machine on every task, but needed to persist within a single cluster of long-lived Windows virtual machine instances. In this case, *HyperShell* server was running under Linux on an HPC cluster front-end, and many clients were coming up asynchronously under Windows.

Resources – *HyperShell* documentation including information for getting started, API reference, and tutorials (work-in-progress) can be found online[1]. The software is now out of beta and the project is available on GitHub[2].

VI. RECUP:A (META)DATA FRAMEWORK FOR REPRODUCING HYBRID WORKFLOWS WITH FAIR

By: Line C. Pouchard, Tanzima Z. Islam, Bogdan Nicolae, and Robert Ross

Successfully reproducing results in computational campaigns that include data-intensive applications and machine learning (ML) is challenging even when the same data input and initial scripts are reused [40], [41]. Large-scale ensemble campaigns where workflow management systems execute tightly coupled task and data processes—so-called hybrid workflows [42] present additional reproducibility challenges due to workflow complexity, scale, distributed execution of tasks, and heterogeneous architectures. The inability to reproduce predictions obtained from ML applications and hence results from hybrid workflows presents a significant obstacle, impairing scientists' ability to validate and trust results; additionally, the lack of reproducibility can inhibit the adoption of research outcomes by others [40]. Findable, Accessible, Interoperable, and Reusable (FAIR) data and workflows can be key enablers in many aspects of scientific discovery [43]. Ensuring the FAIRness of (meta)data (i.e., data and metadata) can

[1]https://hyper-shell.readthedocs.io
[2]https://github.com/glentner/hyper-shell

reduce barriers to reproducibility by making this information easier to find and programmatically access and reuse in new contexts. However, datasets and workflows rarely meet FAIR criteria in the high-performance computing (HPC-ML) context in spite of potential benefits [44]. Identifying the (meta)data required to reproduce results in hybrid workflows has seldom been explored. As a result, significant institutional knowledge is necessary to interpret scientific datasets correctly; complex information must be assembled manually to reproduce workflows; and disparate, platform-specific, heterogeneous sources of performance data must be fused to fully understand runtime performance impeding the rapid progress of scientific discovery. Here, we use the term "reproducibility" in two contexts: **1)** the statistical reproducibility of results and **2)** the reproducibility of computational performance in the execution of workflows, including overhead metrics. Performance reproducibility, defined as achieving the average execution time across multiple runs of the same application using a consistent set of configurations, of such workflows faces additional challenges [45]. Achieving reproducibility requires the ability to collect a history of performance metrics and intermediate results during the runtime of a workflow. This approach allows comparisons of runs from multiple perspectives to identify if they are similar and, if not, when they are diverging and the root cause. In this context, it is essential to capture the evolution of such performance metrics and intermediate results over time and the provenance information that explains how they are derived from each other.

A. A (meta)data framework for reproducing hybrid workflows

Approach – Figure 3 presents a conceptual framework of our approach for identifying, capturing, fusing and storing (meta)data across multiple layers of the computational stack with minimal overhead. We make them FAIR by providing access, publishing metadata in open protocols and repositories, enabling interoperability with existing HPC data and workflow management software, thus facilitating re-use. This innovative conceptual framework introduces multi-modal data fusion (combining data from workflow systems, storage interactions, and task performance along with application information) and provenance services (metadata and associated relationships) for complex and hybrid scientific simulations.

(Meta)data sources – We leverage several composable technologies and services that execute workflows (Radical Cybertools-RCT [46]), extract the provenance of performance at trace level (Chimbuko [47]), and provide data services (Darshan [48], MOCHI [49]). RCT, a standards-based, abstraction-driven toolset for workflow management systems, provides an infrastructure for computational resource provisioning, task scheduling, and scalable execution across HPC systems. Chimbuko is a tool for capturing, analyzing, and presenting in situ performance outliers for extreme-scale workflows. Chimbuko provides detailed, reduced performance measures at a trace level. Darshan is a scalable HPC I/O characterization tool used at LCFs that is the source of filenames and I/O behavior. The Mochi framework provides a methodology and tools for rapidly developing distributed data services.

Result reproducibility using intermediate data states – We extract fine-grain (meta)data from the above sources at key moments during runtime, which enables a comparison of different execution runs and highlighting where the task execution paths diverge, both in terms of performance and result accuracy. Input data from external sources and their metadata annotations (e.g., generated by sensors or reused from a repository) are also considered in the comparison. To this end, we require tools capable of capturing a history of (meta)data snapshots. We leverage two such tools. DataStates [50] is a declarative data access model and middleware that maintains a lineage of snapshots of annotated datasets captured during runtime, making it an ideal candidate to capture rich metadata about performance and intermediate results. VeloC [51] is a multilevel checkpointing runtime for large-scale HPC data centers. While initially designed to enable checkpoint-restart resilience for long-running HPC applications, its versioned, heterogeneous storage-aware capabilities make it an ideal candidate to store intermediate (and potentially large) results as checkpoints.

Performance reproducibility using counter data – For assessing performance reproducibility, we need indicators beyond high-level metrics such as execution time to characterize an application's interactions with the underlying system. For example, analyzing hardware performance counters can expose bottlenecks of various components of a large workflow application. With metadata such as hardware performance counters, directed acyclic graph (DAG), and runtime system metrics, we can address two performance-related research questions– (1) what are the "important" indicators that can explain the difference between run-to-run variation; how important? and (2) how do the performance characteristics differ between two diverging runs of the same application?

B. Future Work

This framework will allow us to envision three types of Fair Digital Objects (FDOs) [52] in the context of hybrid workflows: results FDOs, performance FDOs, intermediate FDOs. These present new challenges to the selection of (meta)data for FAIR re-use and testing with FAIR metrics.

ACKNOWLEDGMENTS

This work is partly funded by NSF award OAC-1839900. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357. libEnsemble was developed as part of the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This research used resources of the OLCF at ORNL, which is supported by the Office of Science of the U.S. DOE under Contract No. DE-AC05-00OR22725.

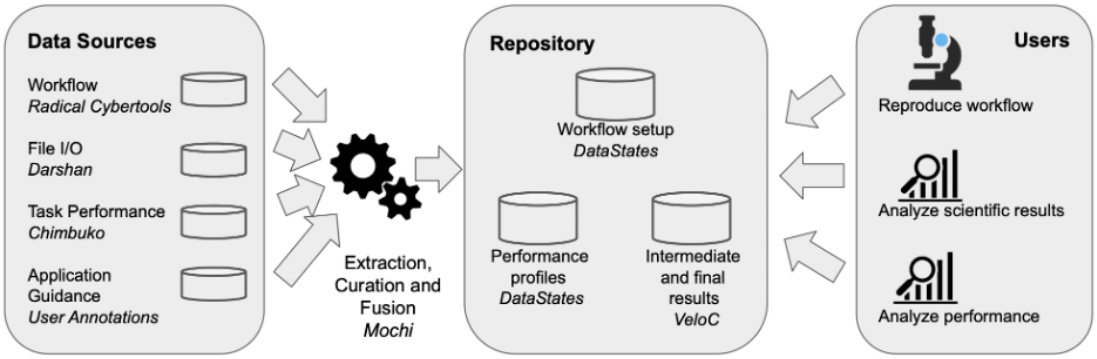

Fig. 3. Multi-modal sources provide data related to I/O behavior, execution of task graph, and task performance.

REFERENCES

[1] R. Ferreira da Silva, H. Casanova, K. Chard, I. Altintas, R. M. Badia, B. Balis, T. a. Coleman, F. Coppens, F. Di Natale, B. Enders, T. Fahringer, R. Filgueira, G. Fursin, D. Garijo, C. Goble, D. Howell, S. Jha, D. S. Katz, D. Laney, U. Leser, M. Malawski, K. Mehta, L. Pottier, J. Ozik, J. L. Peterson, L. Ramakrishnan, S. Soiland-Reyes, D. Thain, and M. Wolf, "A community roadmap for scientific workflows research and development," in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 2021.

[2] S. Peisert, "Security in high-performance computing environments," https : / / cacm . acm . org / magazines / 2017 / 9 / 220422-security-in-high-performance-computing-environments / fulltext, 2017.

[3] M. Rynge, K. Vahi, E. Deelman, A. Mandal, I. Baldin, O. Bhide, R. Heiland, V. Welch, R. Hill, W. L. Poehlman, and F. A. Feltus, "Integrity protection for scientific workflow data: Motivation and initial experiences," in *Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, 2019.

[4] "Open Science Cyber Risk Profile," https://trustedci.github.io/OSCRP, 2022.

[5] "MITRE ATT&CK knowledge base," https://attack.mitre.org, 2022.

[6] I. Abhinit and V. Welch, "Data Integrity Threat Model using Open Science Cyber Risk Profile," https://hdl.handle.net/2022/27980, 2022.

[7] E. K. Adams, "Data Integrity Threat Model using MITRE ATT&CK®," https://hdl.handle.net/2022/28045, 2022.

[8] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, 2015.

[9] L. Lü, M. Medo, C. H. Yeung, Y.-C. Zhang, Z.-K. Zhang, and T. Zhou, "Recommender systems," *Physics reports*, vol. 519, no. 1, 2012.

[10] P. Melville and V. Sindhwani, "Recommender systems." *Encyclopedia of machine learning*, vol. 1, 2010.

[11] "Galaxy main repo," https://usegalaxy.org/workflows/list_published, 2022.

[12] "Galaxy EU repo," https://usegalaxy.eu/workflows/list_published, 2022.

[13] "Galaxy AU repo," https://usegalaxy.org.au/workflows/list_published, 2022.

[14] "myexperiment," https://www.myexperiment.org/workflows, 2022.

[15] "Workflowhub," https://workflowhub.eu/workflows, 2022.

[16] "Dockstore," https://dockstore.org/organizations, 2022.

[17] A. Kumar, H. Rasche, B. Grüning, and R. Backofen, "Tool recommender system in galaxy using deep learning," *GigaScience*, vol. 10, no. 1, 2021.

[18] D. Koop, C. E. Scheidegger, S. P. Callahan, J. Freire, and C. T. Silva, "Viscomplete: Automating suggestions for visualization pipelines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, 2008.

[19] M. M. Junaid, M. Berger, T. Vitvar, K. Plankensteiner, and T. Fahringer, "Workflow composition through design suggestions using design-time provenance information," in *2009 5th IEEE International Conference on E-Science Workshops*, 2009.

[20] "Galaxy main published histories," https://usegalaxy.org/histories/list_published, 2022.

[21] "Galaxy EU published histories," https://usegalaxy.eu/histories/list_published, 2022.

[22] "Galaxy AU published histories," https://usegalaxy.org.au/histories/list_published, 2022.

[23] M. Palmblad, A.-L. Lamprecht, J. Ison, and V. Schwämmle, "Automated workflow composition in mass spectrometry-based proteomics," *Bioinformatics*, vol. 35, no. 4, 2019.

[24] M. DiBernardo, R. Pottinger, and M. Wilkinson, "Semi-automatic web service composition for the life sciences using the biomoby semantic web framework," *Journal of biomedical informatics*, vol. 41, no. 5, 2008.

[25] S. Hudson, J. Larson, J.-L. Navarro, and S. Wild, "libEnsemble: A library to coordinate the concurrent evaluation of dynamic ensembles of calculations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, 2022.

[26] S. Hudson, J. Larson, S. M. Wild, D. Bindel, and J.-L. Navarro, "libEnsemble user manual, version 0.9.2," Argonne National Laboratory, Tech report, 2022. [Online]. Available: https://libensemble.readthedocs.io

[27] A. Ferran Pousa, S. Jalas, M. Kirchen, A. Martinez de la Ossa, M. Thévenet, S. Hudson, J. Larson, A. Huebl, J.-L. Vay, and R. Lehe, "Multitask optimization of laser-plasma accelerators using simulation codes with different fidelities," *Proceedings of the 13th International Particle Accelerator Conference*, no. 13, 2022.

[28] SchedMD, "Slurm workload manager," https://slurm.schedmd.com, 2022.

[29] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, 2011.

[30] G. S. Foundation, "GNU Make," https://www.gnu.org/software/make/, 2022.

[31] F. Mölder, K. P. Jablonski, B. Letcher, M. B. Hall, C. H. Tomkins-Tinch, V. Sochat, J. Forster, S. Lee, S. O. Twardziok, A. Kanitz, A. Wilm, M. Holtgrewe, S. Rahmann, S. Nahnsen, and J. Köster, "Sustainable data analysis with Snakemake," *F1000Research*, vol. 10, no. 9, 2021.

[32] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, and K. Chard, "Parsl: Pervasive parallel programming in Python," in *28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2019.

[33] O. Tange, "GNU Parallel," https://doi.org/10.5281/zenodo.6377950, 2021.

[34] M. G. Grabherr, B. J. Haas, M. Yassour, J. Z. Levin, D. A. Thompson, I. Amit, X. Adiconis, L. Fan, R. Raychowdhury, Q. Zeng, Z. Chen, E. Mauceli, N. Hacohen, A. Gnirke, N. Rhind, F. di Palma, B. W. Birren, C. Nusbaum, K. Lindblad-Toh, N. Friedman, and A. Regev, "Full-length transcriptome assembly from RNA-Seq data without a reference genome," *Nature Biotechnology*, vol. 29, no. 7, 2011.

[35] L. Wilson, J. Fonner, O. Esteban, J. Allison, M. Lerner, and H. Kenya, "Launcher: A simple tool for executing high throughput computing workloads," *Journal of Open Source Software*, 2017.

[36] NERSC, "TaskFarmer," https : / / docs . nersc . gov / jobs / workflow / taskfarmer/, 2022.

[37] G. Lentner and L. Gorenstein, "HyperShell v2: Distributed task execution for HPC," in *Practice and Experience in Advanced Research Computing*, 2022.

[38] *PEARC '22: Practice and Experience in Advanced Research Computing*. Association for Computing Machinery, 2022.

[39] C. Song, P. Smith, X. Zhu, and R. Kalyanam, "NSF award 2005632 - category I: Anvil - a national composable advanced computational resource for the future of science and engineering," https://www.nsf.gov/awardsearch/showAward?AWD_ID=2005632, 2020.

[40] O. E. Gundersen, "The fundamental principles of reproducibility," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 379, no. 2197, 2021.

[41] L. Pouchard, Y. Lin, and H. Van Dam, "Replicating machine learning experiments in materials science," in *Parallel Computing: Technology Trends*, 2020.

[42] R. Filgueira, R. F. Da Silva, A. Krause, E. Deelman, and M. Atkinson, "Asterism: Pegasus and dispel4py hybrid workflows for data-intensive science," in *2016 Seventh International Workshop on Data-Intensive Computing in the Clouds (DataCloud)*, 2016.

[43] K. Fagnan, Y. Nashed, G. Perdue, D. Ratner, A. Shankar, and S. Yoo, "Data and models: a framework for advancing ai in science," USDOE Office of Science (SC)(United States), Tech. Rep., 2019.

[44] 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, 2020.

[45] 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.

[46] 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.

[47] 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.

[48] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," *ACM Transactions on Storage (TOS)*, vol. 7, no. 3, 2011.

[49] 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, 2020.

[50] B. Nicolae, "Datastates: Towards lightweight data models for deep learning," in *Smoky Mountains Computational Sciences and Engineering Conference*, 2020.

[51] B. Nicolae, A. Moody, G. Kosinovsky, K. Mohror, and F. Cappello, "Veloc: Very low overhead checkpointing in the age of exascale," *arXiv preprint arXiv:2103.02131*, 2021.

[52] K. De Smedt, D. Koureas, and P. Wittenburg, "Fair digital objects for science: From data pieces to actionable knowledge units," *Publications*, vol. 8, no. 2, 2020.