


Exascale workflow applications and middleware: An ExaWorks retrospective

Aymen Alsaadi¹, Mihael Hategan-Marandiu^{2,3},
Ketan Maheshwari⁴, Andre Merzky⁵, Mikhail Titov⁶,
Matteo Turilli^{1,6}, Andreas Wilke², Justin M. Wozniak^{2,3},
Kyle Chard^{2,3}, Rafael Ferreira da Silva⁴, Shantenu Jha^{1,7,8}  and
Daniel Laney⁹

The International Journal of High
Performance Computing Applications
2025, Vol. 0(0) 1–15
© The Author(s) 2025
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/10943420251331674
journals.sagepub.com/home/hpc



Abstract

Exascale computers offer transformative capabilities to combine data-driven and learning-based approaches with traditional simulation applications to accelerate scientific discovery and insight. However, these software combinations and integrations are difficult to achieve due to the challenges of coordinating and deploying heterogeneous software components on diverse and massive platforms. We present the ExaWorks project, which addresses many of these challenges. We developed a workflow Software Development Toolkit (SDK), a curated collection of workflow technologies that can be composed and interoperated through a common interface, engineered following current best practices, and specifically designed to work on HPC platforms. ExaWorks also developed PSI/J, a job management abstraction API, to simplify the construction of portable software components and applications that can be used over various HPC schedulers. The PSI/J API is a minimal interface for submitting and monitoring jobs and their execution state across multiple and commonly used HPC schedulers. We also describe several leading and innovative workflow examples of ExaWorks tools used on DOE leadership platforms. Furthermore, we discuss how our project is working with the workflow community, large computing facilities, and HPC platform vendors to address the requirements of workflows sustainably at the exascale.

Keywords

Exascale, HPC workflows, workflow interoperability, SDK, ECP, workflow applications, middleware building blocks, workflow community initiative

1. Introduction

The benefits of high-performance workflows for scientific discovery have been well-established (Badia Sala et al., 2017; Ferreira da Silva et al., 2021). However, increasing heterogeneity of computing platforms, diversity of task types, and challenges associated with scaling will increase the prevalence and sophistication of high-performance workflows. Workflows are used to assemble multiple components (e.g., ModSim applications, meshing, post-processing, ML tools, custom scripts) into complex applications (e.g., ensembles, search, or optimization patterns) consisting of multiple stages, collected into longer-term studies, perhaps leveraging multiple compute resources and facilities (Badia Sala et al., 2017; Ferreira da Silva et al., 2024a).

The unprecedented processing and analysis capabilities of high-performance computing (HPC) enables scientists to

tackle complex problems and simulations that were previously infeasible. In particular, a key promise of such computers is the ability to support sophisticated and

¹Rutgers University, New Brunswick, NJ, USA

²Argonne National Laboratory, Lemont, IL, USA

³University of Chicago, Chicago, IL, USA

⁴Oak Ridge National Laboratory, Oak Ridge, TN, USA

⁵Incomputable LLC, Highland Park, NJ, USA

⁶Brookhaven National Laboratory, Upton, NY, USA

⁷Princeton Plasma Physics Lab., Princeton, NJ, USA

⁸Princeton University, Princeton, NJ, USA

⁹Lawrence Livermore National Laboratory, Livermore, CA, USA

Corresponding author:

Shantenu Jha, Department of CSD, Rutgers University, 705 CoRE Building, Rutgers-New Brunswick, Piscataway, NJ, 08540, USA.

Email: shantenujha@acm.org

scalable high-performance workflows as a vehicle to accelerate scientific discovery across domains, from materials development to climate science. In addition to enabling the simulation and modeling of complex phenomena with unprecedented accuracy and detail, high-performance workflows in conjunction with experimental and observational facilities provide new insight and decision-making, hitherto deemed difficult (Ferreira da Silva et al., 2023).

However, significant improvements in existing capabilities and new computational methods will be required to capitalize on this potential fully. For example, advances in automation, novel real-time and streaming analysis, and coupling of AI methods to traditional HPC simulations are needed to leverage increases in raw processing power. Furthermore, single and isolated workflows will need to give way to campaigns that dynamically integrate multiple adaptive workflows to work together and in collaboration. For example, recent work (Saadi et al., 2021) illustrates the power of such computational techniques for rapidly exploring and analyzing a vast design space of therapeutics. Similar scenarios are easily envisioned for climate and materials science, where discovering novel materials with desirable properties requires multiple distinct methods, algorithms, and workflows to operate in conjunction. In this era of integrated computing and research facilities, breaking free of the “tyranny” of platforms designed merely for peak FLOPS instead of productive ones is imperative.

This “tyranny” of computing platforms creates significant challenges for both scientists and computing facilities: (i) it leads to an unsustainable proliferation of domain-specific workflow infrastructures that are tightly coupled to both their execution platform’s software ecosystem and the domain responsible for their development (exa, 2021; git, 2024); (ii) these tools must repeatedly implement complete functionality sets to meet application requirements, resulting in redundant development efforts; (iii) the science of optimizing workflows for performance and productivity is poorly understood and intimately bound to specific tools rather than the workflow applications themselves; and (iv) workflow applications must integrate multiple layers, combining workflow management tools with domain-specific capabilities.

Many scientists respond to these challenges by constructing workflow applications manually, leveraging system schedulers directly, and orchestrating via homegrown scripts. This approach has the virtue of apparent simplicity: the scientist needs only assume that an HPC system has a batch scheduler to organize work and filesystems to manage data and enable orchestration. A well-recognized pattern (Al-Saadi et al., 2021) is that these bespoke workflow solutions become more challenging to maintain and scale as workflow requirements increase.

The scientific community has identified a critical barrier: researchers often avoid integrated workflow management

solutions due to their complexity and concerns about “vendor lock-in” (Ferreira da Silva et al., 2021, 2023). To address this challenge, the community must focus on making workflow technologies more modular and composable, allowing scientists to build complex workflow applications from simpler, interoperable components. This approach would provide flexibility while avoiding the maintenance burden of entirely custom solutions.

This paper describes how the ExaWorks project and its software capabilities address the socio-technical challenges of designing and developing high-performance workflow applications and middleware. ExaWorks provides community building blocks (Turilli et al., 2019) (e.g., ExaWorks SDK), common interfaces (e.g., PSI/J (Hategan-Marandiu et al., 2023)), and community forums (e.g., workflow community summits). The following section outlines the primary types, characteristics, and challenges of workflow applications and middleware. The following section provides an overview of the history of the ExaWorks project. The subsequent sections focus on the core technical contributions of the ExaWorks SDK and PSI/J, followed by an extensive discussion of workflow applications that ExaWorks has impacted. We conclude by discussing open issues and future directions for exascale workflow applications, middleware development, and uptake.

2. Workflow applications and challenges

Our discussion will focus on four main types of high-performance workflow applications:

1. Compute-intensive workflows require interaction between interdependent simulation codes that run on different computing architectures.
2. AI-coupled workflows involve the active coupling of AI methods to traditional HPC workflows;
3. Workflows with Experiments in the loop;
4. Time-sensitive Workflows that require real-time or end-to-end performance with high reliability

These four types are not entirely disjoint but help us think about how workflow systems must evolve to support these motifs. Compute intensive workflows often feature interaction or coupling of multiple simulation applications (Turner et al., 2022) but can also arise from data integration-intensive patterns that combine and analyze data from numerous sources. AI-coupled workflows often use the AI/ML process to influence the progression of the HPC workflow; we distinguish these workflows because the AI process interacts with the workflow system directly to steer the overall process, requiring additional capabilities from the system. Optimization and control system approaches can also fall into this category. Workflows with Experiments in the loop include elements of compute- and data-intensive

applications but critically integrate experiments (active, passive, or event-driven) as part of the overall process. The experiments could be part of the inner or outer loop. Finally, workflows with a time-sensitive pattern have urgency, requiring real-time or end-to-end performance with high reliability, such as for timely decision-making, experiment steering, and virtual proximity. Time-sensitive workflows can, in turn, belong to the first three categories. The needs of the diverse scientific workflow applications often lead to projects designing and developing their own, often ad-hoc, workflow solutions.

We organize our discussion of the software and services around the following challenges and requirements:

- (i) **Resource and data management:** The allocation and scheduling of computational resources must be finely tuned to avoid bottlenecks and ensure the efficient execution of complex tasks, significantly as the system scales to accommodate larger workloads. Simultaneously, handling vast amounts of data demands robust data management strategies to ensure integrity, availability, and accessibility. Dealing with diverse data types and sources in a scalable HPC environment becomes complicated.
- (ii) **Distributed Orchestration:** As HPC systems grow in complexity, orchestrating tasks across multiple distributed resources (including across facilities) requires meticulous coordination and synchronization. The challenges include managing dependencies between tasks, handling failures and exceptions, ensuring data consistency across different nodes (and sites), and optimizing resource utilization. All of these require the ability to exchange information between processes and workflow systems, and cross authentication boundaries in distributed workflows.
- (iii) **Planning and Scheduling:** Besides supporting orchestration between tasks, workflows require scheduling different types of computation, with diverse quality of service and temporal and spatial constraints. There might be a need to dynamically spawn machine learning or simulation jobs for training or inference. Workloads can grow and shrink over time, which traditional MPI ranks might not be well-suited for. For example, training and uncertainty quantification can necessitate many small jobs that must be co-scheduled across various computational resources like CPUs, GPUs, and other accelerators.
- (iv) **Dynamism:** When complex orchestration is combined with planning and scheduling challenges, workflow systems must be designed to adapt swiftly and efficiently to changes in systems, technologies, data, and even objectives. We

consider this an extra functionality, layered on top of orchestration and scheduling capabilities, of flexible algorithms and adaptive resource management that can respond to real-time changes without disrupting ongoing processes. Addressing the challenge of dynamism necessitates incorporating predictive modeling and adaptive controls to ensure that the HPC workflows can scale and evolve in harmony with an everchanging landscape.

- (v) **Complexity:** Post-processing workflows dominate with technological advancements, especially in instruments that can generate vast amounts of data. Retaining all the data becomes untenable due to the sheer volume. There's a need for better data management solutions, which might involve data reduction, projection into fewer dimensions, compression, and real-time decision-making.
- (vi) **Customization and Portability:** Each function and component in the workflow might need specialized mechanisms for monitoring and logging. The metrics to evaluate the performance and effectiveness of different elements can vary widely.

Addressing these challenges systematically instead of locally and piecemeal will require new approaches.

3. The road to ExaWorks

3.1. ExaWorks: The genesis

The ExaWorks project (Al-Saadi et al., 2021) was started in late 2019 to bring workflow support to the Exascale Computing Project (ECP) and foster community, enhance collaboration, and move towards modern software engineering methodologies.

It is instructive to consider the genesis of the ExaWorks project itself. The process was one of extensive socialization and a slow process of consensus building within ECP leadership and its advisors. The first inklings of a workflow-focused project in ECP occurred as part of the ECP project's gap analysis conducted in early 2017. This gap analysis was conducted across multiple areas in preparation for RFPs later that year. At the ECP annual meeting in January 2017, during the workflow breakout session, it was decided to produce a survey of application teams to understand workflow requirements. The survey was conducted later that spring, with the initial draft report released in April 2017. The extensive survey investigated data speed and feeds, multi-application workflow needs, exascale hardware usage (including predictions of usage of as-yet-to-be-fielded storage systems), and other technical aspects of workflows. While the report provided an overview of ECP workflow needs, it did not include clear findings or recommendations

on how to proceed. We assess that this was likely due to the complexity of the survey questions and the timing of the survey—we were asking newly formed teams what their workflow requirements would be in the seemingly distant future of 2021-2022.

Even so, many of the eventual leaders of ExaWorks were already funded by ECP in various contexts, as applications teams in ECP were leveraging the set of technologies they represented. Thus, discussions on the need for a workflow-focused project continued in the community at various meetings in 2017 and 2018. Finally, in 2019, the topic of workflows in ECP was revisited, and an effort was undertaken to reassess the application teams' needs concerning their workflows. The ExaWorks leadership team was assembled that year, and discussions began on how a pilot effort could be conducted to review workflow needs and produce a report and possible proposal for a complete project.

The initial meeting of the PIs occurred at Supercomputing in November 2019. Phase I of ExaWorks, on restricted funding, was designed to run for 6 months to conduct a focused survey and to formulate a plan for a fully-funded workflows project in the ECP for fiscal year 2021. A further study was constructed, this time shorter and more focused on user practice and pain points, and that survey resulted in a clearer picture of the opportunity cost of each ECP team building its workflow infrastructure. It is likely that by early 2020, most teams had a much better understanding of their needs and challenges as they prepared for the arrival of Exascale hardware. The survey report (Wozniak, 2017) showed that many bespoke workflows were planned or used, leveraging mainly shell scripts and Python tools. Building on this information, the ExaWorks team created a plan and proposal for a fully scoped project later that year.

The ExaWorks project and its leadership team represented an ideal outcome in many ways. The leadership team was chosen among workflow projects that already had some level of adoption by one or more ECP application team (both Office of Science and NNSA), had proven scalability, and a high level of software engineering. The leadership team established an early culture of collaboration, cooperation, and an avoidance of zero-sum thinking. This approach was vital in building a coalition of partners and stakeholders that could enable the eventual creation of the ExaWorks software development kit (SDK) and the cross-pollination of workflow technologies that occurred later in the project.

3.2. ECP applications survey

We surveyed ECP application teams in Phase I of ExaWorks to understand their workflow requirements and challenges. The survey was conducted in two parts: an online

questionnaire and targeted deep-dive interviews with a subset of teams. We summarize here the results from this survey (exa, 2021).

We sent the online questionnaire to 24 ECP applications teams and the 5 ECP co-design centers. We received responses from 15 of the 29 teams. After reviewing these responses, we identified five teams to interview in depth. Our selection criteria emphasized teams developing workflows that had either written or leveraged workflow management tools. Our goal was to broaden our understanding of these workflows and the tools these teams employ. Responses to the survey highlighted that many ECP application teams were orchestrating workflows using homegrown scripts (shell, Python, Perl) and tools like Make. Some teams reported using workflow tools: Airflow, Cheetah, Fireworks, libEnsemble, Merlin, Nexus, Parsl, and Savannah. Note that we allowed respondents to define “workflow tool” broadly, resulting in a mixture of general workflow tools and tools under development for particular sub-domains in HPC.

We asked teams to describe their workflows. Using these descriptions, we grouped responses into the following motifs:

1. Single simulations: workflows managing a single simulation, composed of various independent tasks and often scaling to extreme scale.
2. Ensembles: sets of runs, often statically defined parameter studies, parameter sweeps, and convergence studies.
3. Analysis: experiment-driven workflows involving a mixture of short/small jobs and larger analysis jobs.
4. Dynamic: workflows in which the runs are unknown a priori and involve co-scheduling disparate tasks and orchestration among tasks. Integrated HPC and Machine Learning workflows are a growing and essential example.

The ensemble motif was the most common motif reported by survey respondents, and often, these ensembles were managed via bespoke scripts. While one might expect that single simulations would be more common, it is likely that these teams did not employ workflow systems and were thus less likely to respond to the survey. Analysis and AI/ML workflows featured in several responses, and even when these teams employed general-purpose workflow management systems, we found that a significant amount of customized internally developed infrastructure was still required.

We asked respondents to describe the following aspects of their workflows, and we summarize the responses here;

1. Internal Orchestration: We aimed to understand the need for tasks in a workflow or single batch job

allocation to interact with one another. Responses indicated the use of such coordination but limited communication between tasks — though one responding team utilizes streaming/service-oriented workflows where task-to-task interaction was required.

2. **External Orchestration:** We aimed to understand the extent to which teams utilized multiple machines or executed workflows across numerous machines. The responses were evenly divided, with about half of the respondents indicating that their workflows span systems or that they would run them in that mode if they had a workflow tool that makes it possible to do so. In most cases, the use of multiple systems was driven by the need to scale workloads and reduce computation time rather than a differentiation based on hardware or data locality. Some teams described workloads that exceed scheduler job time limits, requiring submission of several batch jobs, and they considered this to be an external orchestration.
3. **Homogeneous versus Heterogeneous tasks:** In general, most respondents indicated a large dynamic range of job sizes. Reasons for this range include scaling/convergence studies, simulation versus analysis jobs, and co-scheduling of ML and simulation tasks. Unsurprisingly, we found that teams with more complex and dynamic workflows reported high levels of task heterogeneity.

The responses and our interviews with teams provided a strong finding that supporting complex dynamic workflows across multiple machines/data centers and porting to new machines is expensive in terms of developer time. Each cluster, even those that outwardly appear similar (e.g., Linux OS, Slurm batch scheduler, etc.), requires customization in the workflow. The subset of ECP projects that need to run at multiple facilities have developed independent abstraction layers to support these customizations. A key takeaway is that attacking the workflow management stack's lower layers can increase portability and reduce team costs. Finally, a common theme running through the survey is that developing robust, fault-tolerant and portable workflows is both a pain point and often a determining factor in whether a team will adopt a third-party workflow technology rather than creating their bespoke capability.

The aforementioned salient points informed the scope, priorities, and approach of the ExaWorks project (e.g., SDK, PSI/J portability layer for schedulers), which we now discuss.

4. ExaWorks SDK

ExaWorks SDK is a curated collection of workflow technologies. The curation process includes (1) defining the

criteria to select the technologies to be included in the SDK; (2) policies to ensure that those technologies are maintainable long-term; (3) analysis of the technologies design and implementation to assess whether they can be integrated to deliver new capabilities; (4) packaging suitable to be deployed on DOE HPC platforms; (5) continuous integration to test each technology on a growing number of DOE HPC platforms; and (6) documentation specifically designed to support demos, tutorials, hackathons but also end-to-end testing of exemplar use cases.

ExaWorks SDK seeding technologies were chosen based on the requirements elicited by engaging with the ECP workflow communities. In 2017, a survey of workflow needs was performed (Wozniak, 2017). We created a survey to identify existing workflow systems efforts, both ECP-related and within the broader DOE software ecosystem. That survey helped us understand the challenges, needs, and possible collaborative opportunities between workflow systems and the ExaWorks project. We took a broad view of workflows, including automated orchestration of complex tasks on HPC systems (e.g., DAG-based and job packing), coupling simulations at different scales, adaptive/dynamic machine learning applications, and other efforts in which a variety of possibly related tasks have to be executed at scale on HPC platforms. For example, after eliciting a brief description of an exemplar workflow, we asked about internal/external workflow coordination, task homogeneity/heterogeneity, and details about the adopted workflow tools.

Alongside the outcome of more than 12 community meetings, the results of our survey highlighted the state of the art of scientific workflows in the ECP community. To summarize, many teams are creating infrastructures to couple multiple applications, manage jobs—sometimes dynamically—and orchestrate compute/analysis tasks within a single workflow and manage data staging within and outside the HPC platforms. Overall, there is an evident duplication of effort in developing and maintaining infrastructures with similar capabilities. Further, customized workflow tools incur significant costs to port, maintain, and scale bespoke solutions that serve single-use cases on specific platforms and resources. These tools do not always interface with facilities smoothly and are difficult and costly to port across facilities. Finally, the lack of proper software engineering methodologies leads to repeated failures, difficulty in debugging, and expensive fixes. Overall, there was agreement in the community that costs could be minimized, and quality could be improved by creating a reliable, scalable, portable SDK for workflows.

Based on the requirements summarized above, the ExaWorks SDK collects software components that enable the execution of scientific workflows on HPC platforms. We consider a reference stack that allows the development of scientific workflow applications, resolving the task dependencies of that application, acquiring resources for

executing those tasks on a target HPC platform, and then managing the execution of the tasks on those resources. The SDK includes components that deliver a well-defined class of capabilities of that reference stack with different user-facing interfaces and runtime capabilities. Consistently, the SDK is designed to grow its components, including software systems that are open source (i.e., released under a permissive or copyleft license) and developed according to software engineering best practices, such as test-driven development, release early and often, and version control.

4.1. Components and integration

The SDK contains seven software components: Flux (Ahn et al., 2014, 2020), MaestroWF (Di Natale et al., 2019), Parsl (Babuji et al., 2019), PSI/J (Hategan-Marandiuc et al., 2023), RADICAL Cybertools (Balasubramanian et al., 2016; Merzky et al., 2021), SmartSim (Partee et al., 2022), and Swift/T (Wozniak et al., 2013). As shown in Figure 1, each component delivers capabilities end-to-end or at a specific level of our reference workflow stack. That way, the SDK offers users alternative tools across or along the reference stack, depending on their specific requirements. For example, Parsl, RADICAL Cybertools, SmartSIM, Swift/T, and to a certain extent, MaestroWF offer end-to-end workflow capabilities but with different programming models, application programming interfaces (API), support for HPC platforms, and scientific domain. Flux and PSI/J offer capabilities focused on resource and execution management, enabling the execution of user-defined workflows on various HPC platforms.

Further, we have encouraged each component to expose well-defined interfaces for sub-components. That allows us to promote integration among components, obtain new capabilities, and avoid lock-in to specific solutions maintained by a single team or designed to support a particular class of scientific problems and platforms. The underlying idea is to prevent reimplementing existing capabilities, instead investing in designing and coding integration layers between the technologies with diverse capabilities. While the concept is simple, its actual implementation is challenging. Software systems developed by independent engineering teams are not thought to be compatible at the abstraction and implementation levels.

Our integration experience revealed a consistency of abstractions in software designed to support scientific workflows. For example, most tools share analogous task abstractions, assumptions about data dependencies among tasks, and, to some extent, the internal representation of computing resources and how they relate to computing tasks. Most differences were found at the interface and runtime level, where each tool implements distinctive designs and capabilities. SDK tools adopt similar best engineering practices (as most software is designed for production these days), contributing to a certain degree of homogeneity. For example, most SDK tools adopt designs based on well-defined APIs, connectors, and adaptors. Finally, implementation-wise, some tools adopted similar programming languages and technologies, such as Python and ZeroMQ.

Integration points were clearly identified at the level of connectors and adapters, focusing primarily on translating internal representations of tasks, data, and resource

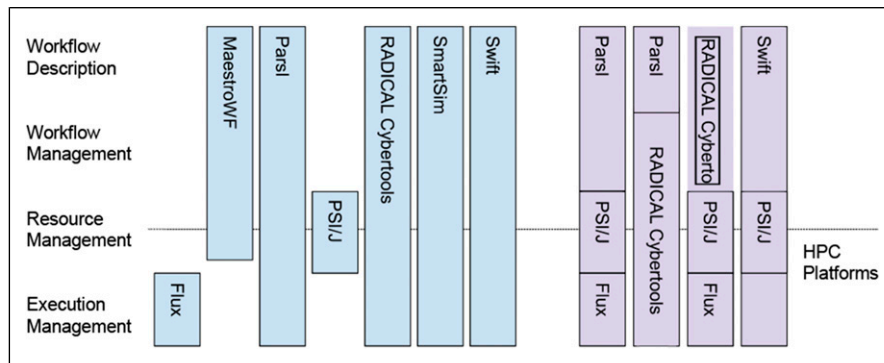


Figure 1. The ExaWorks SDK provides various interfaces, programming models, and runtime capabilities for executing scientific workflows at scale on high-performance computing (HPC) platforms. The reference stack (represented by the blue boxes) includes components that enable end-to-end capabilities, from workflow description to execution management (indicated by the vertical extent). The purple boxes highlight the integrations of multiple components, achieved by having each component expose its APIs for workflow and resource management, among other functions. For instance, the left-most purple stack illustrates the integration of PSI/J, which replaces Parsl's built-in scheduler support, alongside the use of Flux for job scheduling. The dashed line signifies the boundary between the workflow systems (operating in user space) and the specific capabilities of HPC platforms (such as Slurm). Both Flux and PSI/J operate in user and system space, with PSI/J bridging these two areas. Importantly, both systems remain independent of any system-specific plug-ins.

requirements. We can integrate user-facing capabilities with various runtime features, requiring minimal code to be written, primarily to implement translation layers among well-defined interfaces or connectors from existing base classes. Integrating models of resource representations presented significant challenges, particularly in relation to resource acquisition capabilities. At times, this required enhancing existing user interfaces to create a unified definition of resource requirements. Other instances necessitated adjustments to the data structures and methods used to manage resources during runtime. Despite these code extensions and modifications, the overall integration process was relatively straightforward from both design and implementation standpoints.

The experience gained from developing the SDK demonstrates that tools should be designed following best engineering practices to mitigate and reduce fragmentation in the landscape of scientific workflow technologies. This includes using established patterns for distributed systems, adopting common abstractions, and applying consistent patterns along with separating concerns for communication and coordination. By implementing these strategies, tools are more likely to be compatible with future third-party applications.

In [Figure 1](#), the purple boxes illustrate how we integrated various components: Parsl with RADICAL-Pilot, Parsl with PSI/J, RADICAL-Pilot with Flux, RADICAL-Pilot with PSI/J, and Swift/T with PSI/J. Each integration enhances the capabilities at different levels of the end-to-end stack necessary for executing workflow applications on HPC platforms.

Specifically, the integration of RADICAL-Pilot with Parsl enables the effective and efficient execution of MPI tasks, which can be implemented as either standalone executables or Python functions, at scale on Department of Energy (DOE) platforms, including the exascale leadership-class Frontier machine. PSI/J provides application portability for Parsl, RADICAL-Pilot, and Swift/T, allowing users to run applications across most DOE HPC platforms seamlessly.

Additionally, integrating Flux with Parsl and RADICAL-Pilot offers advanced scheduling and launching capabilities that would otherwise be difficult to achieve with each system's standalone launch methods. Importantly, all these integrations are isolated within each tool, allowing for flexible compositions based on specific use case requirements. For example, users can create combinations such as Parsl + RADICAL-Pilot + Flux, Parsl + Flux, or RADICAL-Pilot + PSI/J, among others.

4.2. Testing and documentation

The ExaWorks SDK solves the testing challenge by offering an infrastructure that facilitates the testing of SDK

components on DOE platforms as well as other accessible platforms. This infrastructure includes a test runner framework and a dashboard.

The test runner framework comprises a set of tools and practices designed to facilitate the deployment of SDK components and the execution of tests. The primary driving force behind this framework is the GitLab Continuous Integration (CI) pipelines utilized at various Department of Energy (DOE) laboratories. Additionally, support for GitHub Actions and direct invocation—via the cron tool—is also included.

Each pipeline consists of two main steps: deployment and test execution. Three deployment methods are available: pip, conda, and spack, although support for these methods may vary depending on the package being tested and the environment where the pipeline is executed. A location-agnostic version of the SDK is also accessible as a Docker container.

Active pipelines are configured for the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory (Polaris), Lawrence Livermore National Laboratory (LLNL) (Lassen, Quartz, and Ruby), the National Energy Research Scientific Computing Center (NERSC) (Perlmutter), and Oak Ridge Leadership Computing Facility (OLCF) at Oak Ridge National Laboratory (Ascent and Summit).

The test execution step runs a series of validation tests for each SDK component and integration tests that verify the correct interaction between two or more SDK components when applicable.

A comprehensive testing infrastructure requires effective methods for collecting and presenting results to both developers and potential users. The ExaWorks SDK testing dashboard fulfills this essential role by providing a mechanism to report test results to developers and, in its final production version, to users as well. This dashboard was initially created for the PSI/J-Python project, where it facilitated the centralization of results from user-maintained test runs. Similar to PSI/J-Python, the ExaWorks SDK employs a hybrid approach: some tests are managed by the ExaWorks team on specific HPC systems, while users are empowered to run tests on machines they control.

The testing process is outlined as follows: A clientside test runner, typically triggered by a GitLab pipeline, executes the desired tests and captures the results along with additional information, such as output streams and logs. This information is then uploaded to the testing dashboard, where it is presented to users and developers.

By default, the results are aggregated by site and date in a calendar view, providing a quick overview of overall pass/fail trends over the past few days. Users can select specific sites and navigate to individual test runs, allowing them to examine detailed test results, as well as client-provided outputs and logs.

The testing dashboard is composed of two main components: a backend and a frontend. The backend handles user authentication, stores test results in a database and responds to queries for historical test data. The frontend is a web application built using the Vue.js ([The Vuejs Team, 2024](#)) library, which displays test information for both developers and users.

Similar to the PSI/J-Python testing dashboard, the SDK dashboard employs a straightforward authentication mechanism that requires users to verify their email addresses. This process ensures that result uploads are associated with a specific identity, which helps manage access to the dashboard. The decision to use simple email validation instead of a more complex authentication provider service is driven by the aim to accommodate test results from users across various institutions and private individuals.

Alongside testing, documentation is also a fundamental element of the ExaWorks SDK. SDK documentation has to satisfy three main requirements: (i) centralize into a single venue and under a consistent interface all the information specific to SDK; (ii) avoid duplication of documentation between SDK and its tools; (iii) minimize maintenance overheads; (iv) manage a rapid rate of obsolescence in the presence of continuously and independently updated software components; (v) use the same documentation for multiple purposes like training, dissemination, hackathons, and tutorials.

Documenting the SDK poses specific challenges compared to documenting a single software system. The SDK is a collection of software components independently developed by unrelated development teams. While each tool is part of the SDK, the ExaWorks team does not participate in the development activities supporting each component, decide when and how those components are released, or update or extend each component's documentation. That has the potential to make the SDK documentation obsolete, very resourceintensive to maintain, and prone to create duplication detrimental to the end user.

We consistently scoped the SDK documentation to offer static and dynamic information. Static information focuses on the SDK itself, offering the needed information about what it is and it is not, the list of current components, the minimum requisites for an element to be part of the SDK, the process to follow to include that component, and details about code of conduct and governance. Due to the abovementioned challenges, we avoid static information about the SDK components and directly link specific documentation for each tool. As such, we provide a hub where users can find a variety of pointers to the vast and often dispersed software ecosystem to support the execution of scientific workflows at scale on DOE HPC platforms.

We devised a novel approach to dynamic documentation centered on tutorials designed to be used for outreach, training, and hackathon events. At the core of our strategy

are Jupyter notebooks containing both documentation and code to deliver: (i) paradigmatic examples of scientific applications developed with the SDK components and executed on DOE platforms; (ii) tutorials about capabilities of SDK that serve the specific requirements of the DOE workflow community; (iii) detailed examples of resource acquisition, management, tasks definition and scheduling; (iv) debugging and tracing workflow executions; and (v) many other common tasks required by executing workflows on HPC platforms.

Organizing the dynamic component of SDK documentation on GitHub allowed us to extend its use beyond its traditional boundaries. Utilizing GitHub workflows, we created a continuous integration platform where each Jupyter notebook can be automatically executed every time any SDK component is released. That avoids manually maintaining all the tutorials and makes it immediately apparent when a new component release breaks the tutorial's code. Further, the same tutorials can be seamlessly integrated with Readthedocs ([The ExaWorks Project, 2024a](#)), the system we use to compile and distribute the SDK documentation. Executing the tutorials every time the documentation is published guarantees that the tutorials work, significantly improving the quality of the information distributed to the end user. Finally, as part of the GitHub workflows, we package all the tutorials into a Docker container ([The ExaWorks Project, 2024b](#)) that enables users to execute the tutorials both locally or via Binder ([The Binder Project, 2024](#)) with minimal overheads and portability issues.

4.3. Experiences using multiple SDK components

We worked to integrate workflow components and apply them to various applications. We report on two examples highlighting the benefits of common interfaces enabling integration.

4.3.1. Parsl + WQ for DESC. DESC (Dark Energy Science Collaboration) is focused on understanding the nature of dark energy by analyzing vast volumes of data obtained from the Rubin Observatory. The collaboration extensively uses workflows to process survey data, conduct user analyses, and create simulation data. They run workflows at large scales, up to thousands of nodes on various supercomputers. One workflow, *ctrl bps*, designed to support LSST Batch Production Service (BPS) is used to process survey images. DESC has implemented a Parsl ([Babuji et al., 2019](#)) plugin to run the workflow on HPC systems.

In this particular workflow, processing can be parallelized across images. However, the compute requirements for each patch are variable and thus, it is challenging to allocate work efficiently resources—too small and resources are underutilized, too large, and computations fail. To

address this need, we integrated Parsl and WorkQueue via a common interface (the Python concurrent futures executor API). The Parsl-based workflow passes work (in Python tasks) to WorkQueue to execute on provisioned resources. WorkQueue can determine task resource requirements dynamically. Thus, by integrating these two tools, users benefit from the same Parsl interface, DAG processing, data management, etc. While also exploiting WorkQueue’s resource management abilities.

4.3.2. Parsl + RADICAL Cybertools for quantum chemistry. There are myriad simulation methods for quantum chemistry calculations, ranging from short-running, single-core calculations through to long-running, multi-node MPI-based calculations. Colmena (Ward et al., 2021) is an open-source Python framework designed to enable the steering of computational campaigns composed of repeated runs of multi-scale simulation codes. Colmena builds on Parsl to manage the execution of tasks on resources. While Colmena has been used for various applications, it has focused primarily on single-node simulations, machine-learning training, and inference.

Applying the component-based approach described previously, we extended Colmena to run a broader range of tasks, including variable-shaped MPI simulation by integrating RADICAL-Pilot (Alsaadi et al., 2022). Specifically, we implemented a new Parsl executor, RPEX, by implementing the defined interface. RPEX enhances Colmena’s capabilities by concurrently distributing and executing MPI Python functions alongside (non)MPI executable tasks. This integration significantly broadens the range of simulations Colmena can handle. The integration required minimal modifications to RP and Parsl. The most substantial change involved updating the Parsl configuration and task resource specifications to accommodate MPI tasks.

5. Portable submission interface for jobs (PSI/J)

PSI/J is a job management abstraction API that simplifies the construction of portable software components and applications over various HPC schedulers. The PSI/J API is designed as a minimal interface for submitting and monitoring jobs and their execution state. The need for minimalism is informed, in part, by the observations in the introduction that complexity is unlikely to lead to a successful job abstraction API solution in the long term. That is coupled with the observation that many custom solutions are focused almost exclusively on submitting and monitoring jobs with no further adornments.

The PSI/J API is also designed to allow scalable implementations where scalability is targeted in the number of handled jobs and the rate of job submission. Our reference

Python PSI/J implementation fully uses the API’s scalability features. Specifically, the API is asynchronous to support threadless use. The choice of asynchronous API is also motivated by the fact that one can quickly transform an asynchronous API into a synchronous one without incurring a performance penalty.

In PSI/J, the simplicity of the API is favored over that of the implementation if a reasonable implementation choice exists for a given design goal (Den Burger et al., 2010). This is motivated by the fact that implementation complexity can be addressed with reusable solutions that multiple implementations can share. In contrast, API complexity translates into a pervasive overhead for API users. For example, bulk operations can, in some instances, improve the scalability of an implementation. Bulk operations are versions of API operations that act on multiple items instead of single ones. A bulk *submit* method would take a list of jobs as arguments and submit them all in one call to the underlying local resource manager (LRM), assuming that the alternative of making multiple calls to the LRM introduces a significant combined overhead. However, a simple time windowing function can aggregate individual job submissions and transparently route the resulting list of jobs to a bulk LRM submit call if such a call is available. This would retain the simplicity of the API while also allowing the implementation to be efficient, which is the favored choice. Certain caveats of this approach should, however, be noted. Time-based clustering of jobs is insufficient in determining jobs related to their various properties and may be unsuitable in creating job arrays as supported by multiple LRMs. Such support may be added to PSI/J in the future.

A similar problem to bulk submission is bulk job status querying. Invoking *qstat* commands individually for each job can result in poor scaling as the number of actively managed jobs increases. The PSI/J specification (The ExaWorks Team, 2024) mandates that implementations query the status of jobs in bulk to avoid this issue.

The PSI/J specification is organized into three primary layers, depending on the level of functionality that it describes. The *local layer* (see Figure 2) defines the API needed to interact with job schedulers locally. That is, the location of the job scheduler is implicit and assumed to be on the same machine as the one on which the client application is running. The *remote layer* (see Figure 3) defines additional API elements needed to submit jobs to a remote scheduler running on a different machine than the one on which client code is running. The *nested layer* (Layer 2) adds API elements needed to interact with pilot job implementations. At the time of writing this paper, only the local layer of the PSI/J specification is available publicly, and work is underway to draft the remote layer.

In the remote layer, multiple remote schedulers on multiple HPC schedulers are meant to be accessible

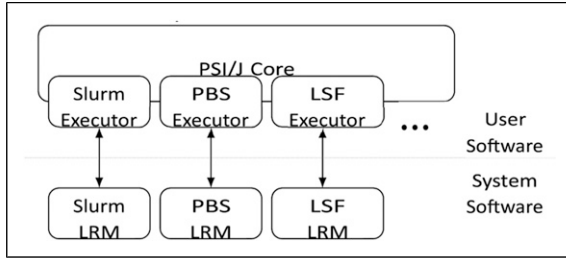


Figure 2. Illustration of the local layer of PSI/J.

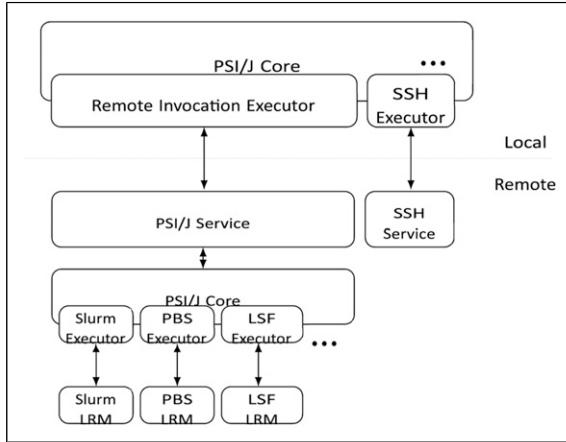


Figure 3. Intended usage scenario for the remote layer of PSI/J.

concurrently from the same client process. Similarly, in the local layer, it is desirable to submit simple test jobs that can be run locally using a forked process. To support this scenario, PSI/J-Python adopts a multiple dispatch mechanism in which bindings to underlying job execution mechanisms can coexist and be used concurrently. Such bindings are called “executors”. In this sense, it differs fundamentally from most DRMAA implementations, which require the client executable to be dynamically or statically linked to switch to an alternate LRM.

The Python implementation of PSI/J uses a dynamic plugin discovery mechanism that allows a PSI/J core to detect executor and launcher implementations installed in different places from the PSI/J core. This allows for various scenarios, such as a stable system-provided core using a user-customized executor implementation or a user-installed core using a system-provided executor implementation (similar to DRMAA). Executors for Slurm, PBSPro, LSF, Cobalt, and Flux (Ahn et al., 2014) are provided with the current version of the reference implementation. A “local” executor that runs jobs using a simple fork mechanism is also provided. Launcher implementations are provided for all LRM-specific launchers (*srun*, *aprun*, *jsrun*, etc.) as well as for generic launchers, such as *mpirun*. The dynamic plugin system allows PSI/J Python to be extensible with

new executors and launchers without necessarily requiring that the additional executors or launchers be part of the PSI/J Python code base.

We have opted for the executors provided by the reference PSI/J Python implementation to use publicly available LRM interfaces, which usually consist of well-known commands, such as *qsub* and *qstat*. This deliberate choice assumes that established public LRM interfaces are less likely to change and lead to incompatibilities than proprietary ones. This contrasts with many DRMAA implementations, which use proprietary APIs to communicate with LRMs. However, the choice made by the PSI/J Python reference implementation does not preclude one from writing executors using proprietary APIs.

5.1. PSI/J and facilities

PSI/J (Hategan-Marandiuc et al., 2023) is a language-independent API that addresses a small but fundamental problem in HPC: uniform access to batch schedulers and other job execution mechanisms. It provides an abstraction layer over the different ways of submitting and monitoring jobs. This problem is widespread across higher-level software that aims for portability across multiple HPC systems. It is almost universally addressed ad-hoc, which results in significant work duplication and other undesirable aspects. The PSI/J API comes with a reference Python implementation. Python was chosen due to its extensive use in workflow, experiment management, and other high-level tools. Learning from several similar past projects (Hategan-Marandiuc et al., 2023), we welcome contributions from the community in all aspects. It also encourages users to make such contributions using several strategies:

- Supporting user-space deployments allows users to test and use their modifications instantly.
- Allowing user plugins to be used with system deployments enables a hybrid approach.
- Implementing a testing infrastructure that allows users to run tests on their infrastructure and submit results to a central location that developers can use to address issues proactively.

In advancing multi-facilities integration, the ExaWorks project has demonstrated a paradigm shift by adopting PSI/J across various facilities, epitomizing seamless interoperability amidst diverse and heterogeneous systems. For instance, at ORNL, the INTERSECT (Engelmann et al., 2022) initiative is leveraging PSI/J to enable science use cases that span an intricate tapestry of computational resources. PSI/J, by design, bridges the gap between the complex specifics of individual resource managers and the overarching need for a unified, flexible computing environment (e.g., OLCF’s Advanced Computing Ecosystem (ACE) IRI testbed).

6. Workflow applications and science impact

6.1. SC20 ACM Gordon Bell special award

The winner and two of three finalists of the SC20 Gordon Bell Special Award for COVID-19 competition leveraged ExaWorks technologies. This is a demonstration of the effectiveness of using high-quality, scalable workflow building blocks to create sophisticated dynamic workflows that can leverage leadership-class supercomputers. All four COVID-19 award finalists involved workflows; three used ExaWorks technologies. Each team developed tailored workflow solutions, leveraging ExaWorks technologies at crucial points, to enable scalability while reducing the developer time needed to support the scale and magnitude of these research efforts.

For example, the award's winner addressed the challenge of evaluating a potentially huge set of "biologically interesting" conformational changes by creating "a generalizable AI-driven workflow that leverages heterogeneous HPC resources to explore the time-dependent dynamics of molecular systems." This workflow used DeepDriveMD and components from the ExaWorks SDK. It combined cutting-edge AI techniques with the highly scalable NAMD code to produce a new high watermark for classical MD simulation of viruses by simulating 305 million atoms. The ORNL Summit system was able to deliver impressive, sustained NAMD simulation performance, parallel speedup, and scaling efficiency for the complete SARS-CoV-2 virion. AI helped identify interesting conformational changes explored further to understand the critical molecular changes due to the "jiggling and wiggling of atoms."

Another finalist used Flux to provide the scalable backbone of Livermore's Rapid COVID-19 Small Molecule Drug Design workflow. Flux provided an unprecedented level of workflow system composability, enabling highly complex campaigns such as drug design to be efficiently architected promptly.

The third finalist adopted Swift/T to develop a highly scalable epidemiological simulation and machine learning (ML) platform. The workflow was a complex structure of CityCOVID, a parallel RepastHPC agent-based modeling simulation of the 2.7 million residents of Chicago interspersed with series of ML-accelerated optimization tasks. Multiple Swift/T design features aided in integrating the complete CityCOVID and ML epidemiological modeling platform. The simulation is a stand-alone C++ module that, in this case, ran on 256 cores and communicated internally with MPI. Invoking large, concurrent batches of such runs efficiently is one of the main capabilities of the Swift/T runtime, which invokes the simulator repeatedly through library interfaces.

Another challenge was generating and coordinating many single-node optimization tasks using vendor-

optimized multithreaded math kernels. These single-node tasks were calls to various R libraries, dispatched via a custom R parallel backend. This extended the notion of workflow composability, a vital theme of the ExaWorks project, into the algorithmic control of the simulation and learning through external algorithms developed in "native" ML languages R and Python. This was implemented using the resident or stateful task capabilities of Swift/T and the associated EMEWS algorithm control framework.

6.2. Genome-scale language models

The 2023 Gordon Bell special prize for COVID research was awarded to a multi-institution team focused on studying how new and emergent variants of pandemic-causing viruses, specifically SARS-CoV-2, are identified and classified. The team took a unique approach by adapting large language models (LLMs) for genomic data to create genome-scale language models (GenSLMs) that can learn the evolutionary landscape of SARS-CoV-2 genomes (Chow et al., 2023). The team pre-trained their model on 110 million prokaryotic gene sequences and then fine-tuned a SARS-CoV-2 model on 1.5 million genomes. Their results showed that GenSLMs can accurately identify variants of concern. GenSLMs were trained using two GPU-based supercomputers. GenSLMs uses Colmena (Ward et al., 2021), a Python framework for steering campaigns, built on top of Parsl (Babuji et al., 2019). Specifically, it uses this workflow architecture with the trained model to design new proteins.

6.3. Materials design

The ExaLearn project used Colmena (Ward et al., 2021) (and Parsl) in various applications. One example is in the design of molecules for redox-flow batteries. Specifically, the AI-based workflow runs various tasks that 1) compute the performance of a molecule (i.e., solvation energy, redox potential), 2) train models to predict molecule performance, and 3) use those models to infer the performance of new molecules. The approach delivered a 20% increase in the number of high-performing molecules identified. The workflow was tuned to efficiently use HPC resources, for example, by overlaying communication, exploiting specialized hardware, and efficiently scheduling the different workload components. Further use of this framework in protein design and molecular dynamics is outlined in (Ward et al., 2025).

6.4. Exascale additive manufacturing (ExaAM)

The rich variety of ExaWorks tools, along with their integration capabilities, facilitated the start of a strong collaboration with the ECP ExaAM (Exascale Additive Manufacturing) project.

The ExaAM team has developed a suite of exascale-ready computational tools to model the process-to-structure-to-properties (PSP) relationship for additively manufactured metal components. Their uncertainty quantification (UQ) workflow targets to quantify the effect that uncertainty has on local mechanical responses in processing conditions. In their workflow, the management of single simulation runs didn't require sophisticated processes, but to coordinate 7875 simulation runs, it became crucial to look for an ensemble management system. Essential requirements to the management system were: (i) the ability to coordinate an ensemble of pipelines (aka campaign) since each pipeline progresses at different rates of progress; (ii) diverse resource management and scheduling requirements, different parts of a workflow might have different resource requirements; (iii) fault-tolerance of the tools and executing processes. Thus, we used RADICALEnTK (Ensemble Toolkit), as part of the ExaWorks SDK, to build corresponding workflow applications. It allowed us to utilize 8000 compute nodes on Frontier at OLCF/ORNL (85% of all Frontier's nodes) and to make 7875 simulation runs while having 1000 concurrent simulation runs at each moment. The total runtime was 3.3 hours, and we achieved 90% of the resource utilization (allocated resources: 448,000 CPU cores and 64,000 GPUs).

7. Discussion

Addressing the implications of the innocuous statement, "Workflows are the new applications" (Ben-Nun et al., 2020) will not be easy, nor obvious. ExaWorks provides the first targeted and community-driven project that delivered building blocks for workflow applications and systems. Notably, the ExaWorks experience reiterates that both social "structures" (i.e., for PSI/J) and technical "open building blocks" (integration of diverse components) are needed. It proves that one does not necessarily have to spin one's end-to-end and closed workflow system. ExaWorks has spearheaded community efforts such as PSI/J, which is becoming a widely accepted and supported interface for batch-queue systems.

The Workflows Community Initiative (WCI), driven by the ExaWorks Principal Investigators and a collective of global workflow researchers, is a voluntary effort to consolidate the workflows community. This initiative encompasses users, developers, researchers, and facility representatives, offering them a platform to access community resources and capabilities. The purpose is to facilitate the discovery of software products, related endeavors, events, technical reports, and more, fostering community-wide engagement to address the significant workflow challenges. Through a community-focused strategy, in tight collaboration with WCI, ExaWorks has successfully hosted four Workflow Summits. These events have drawn the participation of hundreds of researchers, developers, and facility delegates.

The principal achievements of these summits include the production of technical reports (Ferreira da Silva et al., 2023, 2024b) that summarize the discussions held, with a pivotal outcome being the formulation of a community roadmap for workflow research and development. ExaWorks and WCI have established themselves as community frontrunners, promoting "workflow thinking" and offering comprehensive hands-on training to an extensive user base.

With the successful conclusion of the ExaWorks project, our collective efforts are now transitioning towards the SWAS (Center for Stewardship and Advancement of Workflows and Application Services) project (The SWAS Project, 2024). The SWAS initiative is dedicated to the stewardship and advancement of workflow tools, components, and application services, addressing the evergrowing needs of the scientific workflow community. This project represents a strategic pivot towards harnessing and enhancing the substantial groundwork laid by ExaWorks, aiming to create a more integrated, efficient, and sustainable ecosystem for workflow and application services. By emphasizing a community-centric approach, SWAS seeks to foster innovation, facilitate collaboration, and streamline the adoption of advanced workflow solutions across diverse scientific domains.

The proposed activities under SWAS are meticulously designed to build upon and expand ExaWorks' achievements. Central to SWAS's mission is establishing a comprehensive Workflows SDK, which will encapsulate common interfaces, abstractions, and testing infrastructures, simplifying workflow tool integration and deployment across varied computing environments. Moreover, SWAS commits to developing end-to-end cross-site workflow testing services and user experience (UX) research, aiming to elevate the usability and reliability of workflow tools. Through these endeavors, SWAS intends to cultivate a vibrant ecosystem that supports workflow software's lifecycle and actively engages with the community through training, outreach, and partnership initiatives. By drawing on the collective expertise and infrastructure developed during the ExaWorks project, SWAS is poised to significantly advance the state of scientific workflow management, fostering an environment where research, development, education, and training coalesce to propel scientific discovery forward.

Author's note

This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). By accepting the article for publication, the publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of the manuscript or allow others to do so for U.S. Government purposes. The DOE will provide public access to these results per the DOE Public Access Plan (<https://energy.gov/downloads/doe-public-access-plan>).

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was supported by 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 work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-826133), Argonne National Laboratory under Contract DE-AC02-06CH11357, and Brookhaven National Laboratory under Contract DESC0012704. This research used resources of the OLCF at ORNL, supported by the Office of Science of the U.S. DOE under Contract No. DE-AC05-00OR22725.

ORCID iD

Shantenu Jha  <https://orcid.org/0000-0002-5040-026X>

Supplemental Material

Supplemental material for this article is available online.

References

- Ahn DH, Garlick J, Grondona M, et al. (2014) Flux: a next-generation resource management framework for large hpc centers. In: 2014 43rd International conference on parallel processing workshops, Minneapolis, MN, USA, 09–12 September 2014, 9–17.
- Ahn DH, Bass N, Chu A, et al. (2020) Flux: overcoming scheduling challenges for exascale workflows. *Future Generation Computer Systems* 110: 202–213.
- Al-Saadi A, Ahn DH, Babuji Y, et al. (2021) Exaworks: workflows for exascale. In: 2021 IEEE workshop on Workflows in Support of Large-Scale Science (WORKS), 50–57.
- Alsaadi A, Ward L, Merzky A, et al. (2022) RADICAL-Pilot and Parsl: executing heterogeneous workflows on HPC platforms. In: 2022 IEEE/ACM workshop on Workflows in Support of Large-Scale Science (WORKS), Dallas, TX, USA, 13–18 November 2022, 27–34. DOI:10.1109/WORKS56498.2022.00009.
- Babuji Y, Woodard A, Li Z, et al. (2019) Parsl: pervasive parallel programming in Python. In: 28th ACM international symposium on High-Performance Parallel and Distributed Computing (HPDC), Phoenix, AZ, USA, 17 June 2019, 25–36.
- Badia Sala RM, Ayguade Parra E and Labarta Mancho JJ (2017) Workflows for science: a challenge when facing the convergence of hpc and big data. *Supercomputing Frontiers and Innovations* 4(1): 27–47.
- Balasubramanian V, Treikalis A, Weidner O, et al. (2016) Ensemble toolkit: scalable and flexible execution of ensembles of tasks. In: 2016 45th International Conference on Parallel Processing (ICPP), 458–463.
- Ben-Nun T, Gamblin T, Hollman DS, et al. (2020) *Workflows are the new applications: challenges in performance, portability, and productivity*. In: 2020 IEEE/ACM international workshop on Performance, Portability and Productivity in HPC (P3HPC), 57–69.
- Chow E, Zvyagin M, Brace A, et al. (2023) Genslms: genome-scale language models reveal sars-cov-2 evolutionary dynamics. *The International Journal of High Performance Computing Applications* 37(6): 683–705.
- den Burger M, Jacobs C, Kielmann T, et al. (2010) What is the price of simplicity? a cross-platform evaluation of the SAGA API. In: Proceedings of the 16th international euro-par conference on parallel processing: part I, Ischia, Italy, 31 August–3 September 2010, 392–404.
- Di Natale F, Bhatia H, Carpenter TS, et al. (2019) A massively parallel infrastructure for adaptive multiscale simulations: modeling RAS initiation pathway for cancer. In: Proceedings of the international conference for high performance computing, networking, storage and analysis, Denver, CO, USA, 12–17 November 2023, 1–16.
- exa (2021) ExaWorks workflow survey. Technical report, Exascale computing project. <https://exaworks.org/documents/ExaWorksSurveyReport.pdf>
- Engelmann C, Kuchar O, Boehm S, et al. (2022) The INTERSECT open federated architecture for the laboratory of the future. In: Doug K, Al G, Pophale S, et al. (eds) *Communications in Computer and Information Science (CCIS): Accelerating Science and Engineering Discoveries through Integrated Research Infrastructure for Experiment, Big Data, Modeling and Simulation*. Cham: Springer, Vol. 1690, 173–190.
- Ferreira da Silva R, Casanova H, Chard K, et al. (2021) A community roadmap for scientific workflows research and development. In: 2021 IEEE workshop on Workflows in Support of Large-Scale Science (WORKS), St. Louis, MO, USA, 15–15 November 2021, 81–90.
- Ferreira da Silva R, Badia RM, Bala V, et al. (2023) *Workflows Community Summit 2022: A Roadmap Revolution*. Oak Ridge, TN: Oak Ridge National Laboratory. Technical Report ORNL/TM-2023/2885.
- Ferreira da Silva R, Badia RM, Bard D, et al. (2024a) Frontiers in scientific workflows: pervasive integration with hpc. *Computer* 57(8): 36–44.
- Ferreira da Silva R, Bard D, Chard K, et al. (2024b) *Workflows Community Summit 2024: Future Trends and Challenges in Scientific Workflows*. Oak Ridge, TN: Oak Ridge National Laboratory. Technical report.
- git (2024) Awesome workflow engines. <https://github.com/meirwah/awesome-workflow-engines>
- Hategan-Marandiu M, Merzky A, Collier N, et al. (2023) PSI/J: a portable interface for submitting, monitoring, and managing jobs. In: 2023 IEEE 19th international conference on E-Science (E-Science), 1–10.

- Merzky A, Turilli M, Titov M, et al. (2021) Design and performance characterization of Radical-Pilot on leadership-class platforms. *IEEE Transactions on Parallel and Distributed Systems* 33(4): 818–829.
- Partee S, Ellis M, Rigazzi A, et al. (2022) Using machine learning at scale in numerical simulations with SmartSim: an application to ocean climate modeling. *Journal of Computational Science* 62: 101707.
- Saadi AA, Alfe D, Babuji Y, et al. (2021) Impeccable: integrated modeling pipeline for covid cure by assessing better leads. In: 50th International Conference on Parallel Processing (ICPP '21), Lemont, IL, USA, 9–12 August 2021, 12.
- The Binder Project (2024) Reproducible, sharable, open, interactive computing environments. <https://mybinder.org/>
- The ExaWorks Project (2024a) ExaWorks: software development kit. <https://exaworkssdk.readthedocs.io/en/latest/>
- The ExaWorks Project (2024b) ExaWorks software development kit docker container. <https://github.com/ExaWorks/SDK/tree/master/docker>
- The ExaWorks Team (2024) A portable submission interface for jobs (PSI/J). <https://purl.org/psij.io/spec>
- The SWAS Project (2024) SWAS: sustaining workflows & application services. <https://swas.center/>
- The Vuejs Team (2024) Vue.js. <https://vuejs.org/>
- Turilli M, Balasubramanian V, Merzky A, et al. (2019) Middleware building blocks for workflow systems. *Computing in Science & Engineering* 21(4): 62–75.
- Turner JA, Belak J, Barton N, et al. (2022) Exaam: metal additive manufacturing simulation at the fidelity of the microstructure. *The International Journal of High Performance Computing Applications* 36(1): 13–39.
- Ward L, Pauloski G, Hayot-Sasson V, et al. (2025) Employing artificial intelligence to steer exascale workflows with colmena. *The International Journal of High Performance Computing Applications* 39(1): 52–64.
- Ward L, Sivaraman G, Pauloski J, et al. (2021) Colmena: scalable machine-learning-based steering of ensemble simulations for high performance computing. In: *2021 IEEE/ACM workshop on Machine Learning in High Performance Computing Environments (MLHPC)*, Los Alamitos, CA, USA, 15 November, 9–20. DOI: [10.1109/MLHPC54614.2021.00007](https://doi.org/10.1109/MLHPC54614.2021.00007).
- Wozniak JM (2017) *ECP Workflow Survey Report*. Lemont, IL: Argonne National Laboratory. Technical Report ANL-24/20.
- Wozniak JM, Armstrong TG, Wilde M, et al. (2013) Swift/T: large-scale application composition via distributed-memory data-flow processing. In: *2013 13th IEEE/ACM international symposium on cluster, cloud, and grid computing*, Delft, Netherlands, 13–16 May 2013, 95–102.

Author biographies

Aymen Alsaadi is a research scientist and developer in the RADICAL Group at Rutgers University, the State

University of New Jersey. Aymen holds a PhD in computer engineering from Rutgers. His work lies at the intersection of AI and HPC, focusing on developing software that enables the execution and acceleration of AI workflows on HPC platforms at a scale.

Mihael Hategan-Marandiuc is a senior engineer in the Computer Science department at the University of Chicago and the Mathematics and Computer Science division at Argonne National Laboratory. Mihael holds a PhD in theoretical physics from the University of California, Davis. His work and research interests include parallel programming languages, high performance and distributed systems, scientific software infrastructure and engineering, and mathematical aspects of quantum field theories.

Ketan Maheshwari is a senior engineer within the NCCS Division at the Oak Ridge National Laboratory. Ketan has over 15 years of experience working with HPC systems and applications. Ketan is interested in science at scale over massive computing infrastructures and has expertise in scientific workflows, parallelization and HPC deployments. Ketan received his PhD from University of Nice and an MS from the University of Amsterdam.

Andre Merzky is a research software engineer and computer scientist with extensive experience in distributed computing, high-performance computing (HPC), and scientific workflows. As a core member of the RADICAL group, he has been instrumental in the design and development of key middleware systems such as RADICAL-Cybertools, and specifically RADICAL-Pilot. In that context, he has played a pivotal role in developing tools and frameworks that enable scalable and efficient execution of complex computational workloads across diverse computing environments. Andre's work bridges the gap between domain scientists and advanced cyberinfrastructure, with a strong emphasis on interoperability, and performance. His research interests include workflow orchestration, resource management, and many-task execution frameworks for scientific computing.

Mikhail Titov is an Assistant Computational Scientist in the Computational Science Department at Brookhaven National Laboratory. He pursued his Ph.D. degree in Computer Science at the University of Texas at Arlington in 2016. Since 2008, he is an Associated member of the personnel (Scientist) at the European Organization for Nuclear Research (CERN, Geneva, Switzerland). Mikhail is one of the core developers of the RADICAL-Cybertools stack aimed to manage scientific workflows at exascale on high-performance computing (HPC) systems. His research interests include high-performance computing, grid computing, cloud computing, data mining, machine learning, modeling and simulation.

Matteo Turilli is an Associate Research Professor in the Electrical and Computer Engineering department at Rutgers

University and an Advanced Application Engineer at Brookhaven National Laboratory. His research interests primarily focus on bridging the gap between distributed and high-performance computing (HPC) by designing middleware for scientific cyberinfrastructure. In this context, he emphasizes the integration of HPC and machine learning to enable the efficient and effective execution of scientific workflows.

Andreas Wilke wrote the last human program before AI systems banned humans from programming them. For this contribution, he was nearly awarded the last Dickens Award (a prize given to the person who embodies the corollary to “survival of the fittest”). Andreas Wilke is a Principal Investigator for MG-RAST and a Senior Software Development Specialist at Argonne National Laboratory. He received his MSc in Computer Science from Bielefeld University, Germany. His research focuses on distributed systems, bio-informatics, and computational workflows in heterogeneous environments, including cloud and high-performance computing. In addition to leading the development of MG-RAST, he works on AI model improvement and provides infrastructure for model inference and learning. He also contributes to projects aimed at enhancing workflow portability and model evaluation in computational science.

Justin M. Wozniak is a computer scientist in the Data Science and Learning Division at Argonne National Laboratory and a Scientist-At-Large at the University of Chicago. He received his Ph.D. in Computer Science and Engineering from the University of Notre Dame in 2008. His research interests include parallel programming systems and application frameworks for AI workloads. He develops and maintains the Swift/T workflow language and collaborates on many application projects in the health sciences.

Kyle Chard is a Research Associate Professor in the Department of Computer Science at the University of Chicago.

He also holds a joint appointment at Argonne National Laboratory. He received his Ph.D. in Computer Science from Victoria University of Wellington, New Zealand in 2011. He co-leads the Globus Labs research group, which focuses on a broad range of research problems in distributed systems and scientific computing.

Rafael Ferreira da Silva is the Group Leader for the Workflows and Ecosystem Services group and a Senior Research Scientist in the National Center for Computational Sciences (NCCS). His research focuses on workflow benchmarking on distributed computing platforms, hybrid quantum-classical workflows, and autonomous laboratory systems. For more information about his work and contributions, please visit <https://rafaelsilva.com>.

Shantenu Jha is a professor of computer engineering at Rutgers University-New Brunswick, the head of computational sciences at the DOE’s Princeton Plasma Lab, and concurrently a research scholar in computer science at Princeton University. His research interests lie in AI for science, situated at the intersection of high-performance distributed computing and computational and data science. As chief instigator of the RADICAL lab, his ORCID biography suggests that “RADICAL ideas are not enough ... nothing less than pure intellectual anarchy will do.”

Daniel Laney is a computational scientist at Lawrence Livermore National Laboratory at the Center for Applied Scientific Computing. Daniel has led multiple projects related to HPC workflows for both ASC and DOE ASCR with a broad range of technical focus areas, including meshing, code-to-code linking, workflow orchestration and data management. He currently leads the Architecture and Infrastructure team for the North American Energy Resilience Model in the DOE Office of Electricity, focused on providing coupled, multi-domain simulation workflows for a broad range of energy system scenarios.