BNL-222941-2022-JAAM

ExaWorks: Workflows for Exascale

A. Al-Saadi, M. Titov

To be published in "PROCEEDINGS OF 16TH WORKSHOP ON WORKFLOWS IN SUPPORT OF LARGE-SCALE SCIENCE (WORKS21)"

November 2021

Computational Science Initiative

Brookhaven National Laboratory

U.S. Department of Energy

USDOE Office of Science (SC), Basic Energy Sciences (BES) (SC-22)

DISCLAIMER

ExaWorks: Workflows for Exascale

Aymen Al-Saadi[1], Dong H. Ahn[2], Yadu Babuji[3,4], Kyle Chard[3,4], James Corbett[2],
Mihael Hategan[3,4], Stephen Herbein[2], Shantenu Jha[5,1], Daniel Laney[2], Andre Merzky[5],
Todd Munson[3], Michael Salim[3], Mikhail Titov[5], Matteo Turilli[1,5], Thomas D. Uram[3], Justin M. Wozniak[3]
[1] *Rutgers, the State University of New Jersey, Piscataway, NJ 08854, USA*
[2] *Lawrence Livermore National Laboratory, Livermore, CA 94550, USA*
[3] *Argonne National Laboratory, Lemont, IL 60439, USA*
[4] *The University of Chicago, Chicago, IL 60637, USA*
[5] *Brookhaven National Laboratory, Upton, NY 11973, USA*

***Abstract*—Exascale computers will offer transformative capabilities to combine data-driven and learning-based approaches with traditional simulation applications to accelerate scientific discovery and insight. These software combinations and integrations, however, are difficult to achieve due to challenges of coordination and deployment of heterogeneous software components on diverse and massive platforms. We present the ExaWorks project, which can address many of these challenges: ExaWorks is leading a co-design process to create a workflow Software Development Toolkit (SDK) consisting of a wide range of workflow management tools that can be composed and interoperate through common interfaces. We describe the initial set of tools and interfaces supported by the SDK, efforts to make them easier to apply to complex science challenges, and examples of their application to exemplar cases. Furthermore, we discuss how our project is working with the workflows community, large computing facilities as well as HPC platform vendors to sustainably address the requirements of workflows at the exascale.**

I. Introduction

The coupling of traditional High Performance Computing (HPC) with new simulation, analysis, and data science approaches provides unprecedented opportunities for discovery but also creates new application and infrastructure challenges. Several Exascale Computing Project (ECP) [1] workflows exemplify this new reality [2], [3]: a heterogeneous combination of applications, Machine Learning (ML) models, and "glue" code, running on heterogeneous compute nodes, orchestrated by a scalable workflow system. These workflows require specialized workflow management software which are currently available to only certain large and specialized inter-disciplinary teams. The gap between capability and requirements will become more acute with scale and sophistication. Furthermore, "bespoke" approaches to workflow development have resulted in many inflexible, tightly integrated, and stove-piped software solutions. An explosion in the number of independent solutions is making development and support of workflows increasingly unwieldy, expensive and unsustainable.

Several technical and non-technical challenges impede the creation of portable, repeatable, and performant workflows. On the technical side, workflow management systems (WMS) and complex workflows are difficult to port and maintain which hinders usability, portability and ultimately adoption. On the non-technical side, myriad WMS exist which often try to provide complete and end-to-end capabilities, resulting in duplicated general capabilities but missing specialized functionality. The lack of coordination, high-quality specialized and broadly usable common components, has resulted in a disjoint workflows community that tends towards building ad hoc solutions rather than adopting and extending existing solutions. The complex workflows landscape demands an open community-based approach to address these challenges.

The ECP ExaWorks project was created in response to these challenges and is addressing both technical and non-technical aspects. We are co-designing the ExaWorks SDK comprised of scalable workflow tools that can be combined to enable diverse teams to produce scalable and portable workflows for exascale applications. We do *not* aim to replace the many workflow solutions already deployed and used by scientists, but rather to provide a well-defined and scalable SDK that provides both common and interoperable APIs, as well as well-tested workflow technologies, to both the user and workflow communities. Most importantly, the SDK will enable sustainability via the creation of a continuous integration and deployment (CI/CD) infrastructure so that software artifacts produced by participating teams will be easier to port and maintain. SDK components will be usable by other WMSs thus facilitating software convergence in the workflows community. The ExaWorks SDK is intended to provide scalable technologies while moving towards sustainability, re-usability, and adoption:

1) **Re-usability and Composability:** we are partnering with the workflow community to define natural integration points between workflow technologies and begin to define common APIs and reference implementations for capabilities implemented in many workflow systems.

2) **Sustainability:** the ExaWorks SDK will be included in the Extreme Scale Software Stack (E4S) [4], a community effort to provide open-source software packages for developing, deploying and running scientific applications on HPC platforms. E4S provides from-source builds and containers of a broad collection of HPC software packages.

3) **Adoption:** we are building comprehensive SDK documentation, with user-facing examples and tutorials, that will facilitate adoption of workflow technologies by developers.

ExaWorks is also a community-driven project. Our vision is to create an open process and community *curated* SDK for workflows: first define interfaces for logical workflow com-

ponents and then bootstrap the ExaWorks SDK by adapting a set of existing WMS components currently being leveraged by ECP applications.

In this paper, we present the initial set of ExaWorks technologies and PSI/J, a first component API that aims to provide a unified interface to job schedulers. We then highlight examples of cross-integrations among the current SDK technologies and provide highlights from recent application of ExaWorks technologies to extreme-scale workflows.

II. Understanding HPC Workflows

Before embarking on the ExaWorks project, we conducted a survey of ECP application teams 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. In this section we summarize the results and takeaways from this survey [5].

We sent the online questionnaire to 24 ECP applications teams and the 5 ECP co-design centers. We received responses from 15 out of the 29 teams. After reviewing these responses we identified five teams to interview in depth. Our selection criteria emphasized teams that were developing workflows and that had either written or were leveraging workflow management tools. Our goal here was to broaden our understanding of these workflows and the tools employed by these teams.

Responses to the survey highlighted that many ECP application teams are orchestrating workflows using homegrown scripts (shell, Python, Perl) and tools like Make. Some teams reported usage of workflow tools: Airflow, Cheetah, Fireworks, libEnsemble, Merlin, Nexus, Parsl, and Savannah. Note, 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 which involve a mixture of short/small jobs and larger analysis jobs.
4) **Dynamic:** workflows in which the runs are not known a priori and that involve co-scheduling of disparate tasks and orchestration among tasks. Integrated HPC and Machine Learning workflows are a growing and important 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 ML/dynamic workflows featured in several responses, and even when general purpose workflow management systems were employed by these teams 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 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 multiple 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 to 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 as a case of external orchestration.
3) **Homogeneous vs. Heterogeneous tasks:** In general, most respondents indicated a large dynamic range of job sizes. Reasons for this range include: scaling/convergence studies, simulation vs. 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.), require 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 lower layers of the workflow management stack can bring increased portability and reduce costs for teams. Finally, a common theme running through the survey is that developing robust workflows that are fault tolerant and portable is both a pain-point and oftentimes a determining factor in whether a team will adopt a third party workflow technology rather than creating their own bespoke capability.

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

III. Three Components of ExaWorks

The three pillars of the ExaWorks technical approach are the robust and performant component technologies, APIs for common workflow components, and the assembly of an open SDK. We describe each of these pillars below.

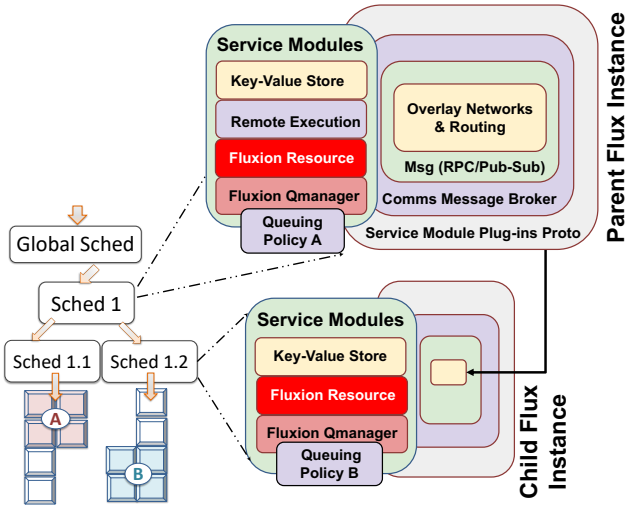


Fig. 1. Flux's fully hierarchical software framework architecture

A. Exaworks Components Technologies

Balsam [6] provides a hosted platform for orchestrating distributed workflows via web-accessible APIs or Python SDK. Users define and manage a collection of HPC execution sites that automate wide-area data transfers, resource allocation, and high-throughput, fault-tolerant execution of tasks via pilot jobs. Since Balsam sites run as user-domain clients communicating with the service over HTTPS, secure deployments are straightforward across diverse platforms. For instance, Balsam sites can be installed and run on systems ranging from laptops to DOE supercomputers. Armed with a collection of sites, clients may then remotely submit tasks to the Balsam API to distribute workloads in near-real time computing scenarios.

The Balsam site's user agent is architected as a collection of platform-agnostic modules (auto-scaling, resource manager synchronization, pilot jobs, data staging) that interface with the underlying systems through a set of adapters implementing *platform interfaces* (Batch Scheduler, MPI Launcher, Compute Resource, Data Transfer Protocol). Balsam sites can be deployed to new systems by setting the appropriate adapters, while supporting new HPC schedulers or application launch paradigms requires minimal implementation of well-defined interfaces. Pilot jobs dynamically pull workloads from the API and schedule execution of heterogeneous tasks across available resources. The service maintains a history of state transitions for each task, enabling users to register *pre-/post-processing* or *timeout-/error-handling* hooks, which are invoked by the Balsam site at appropriate stages of the task lifecycle.

Flux [7] is a fully hierarchical workload manager for HPC. It was born out of growing computing needs for more sophisticated scheduling and resource management of larger, more heterogeneous and dynamic systems at facilities such as DOE national laboratories. Its fully hierarchical capabilities have proven to improve scalability and flexibility significantly through a divide-and-conquer approach that is well-suited for emerging environments. Jobs and resources are divided among the Flux instances in the hierarchy and managed in parallel.

Fig. 1. shows the modular architecture of Flux, and also depicts how its network can be organized to manage two Flux instances at different levels of the hierarchy, with a parent Flux instance and a child Flux instance. The hierarchical design of Flux provides ample parallelism and flexibility to overcome emerging workflow challenges (e.g., higher job throughput requirement). Under the hierarchical design of Flux, any Flux instance can spawn child instances to aid in scheduling, launching, and managing jobs. The parent Flux instance grants a subset of its jobs and resources to each child. This parent-child relationship can extend to an arbitrary depth and width, creating many opportunities for parallelization and drastically increasing the scalability of Flux over traditional schedulers that rely on a centralized scheme. In addition, Flux uniquely provides two modes of operations: single- and multi-user modes. Emerging scientific workflows often leverage its single-user mode whereby hierarchical workload management is provided in *user space* within a system-level batch allocation created by HPC system workload managers, such as Slurm and IBM LSF. Such a workload-management overlay allows users to set up their own customized hierarchies and tune scheduling policies tailored to their workflow.

Parsl [8] is a parallel programming library for Python. Parsl augments standard Python with workflow constructs to define dataflow and control semantics. Parsl requires that individual workflow components be implemented as Parsl *Apps*–annotated Python functions that wrap pure Python code or Bash commands. By annotating these functions, Parsl knows that they can be executed concurrently. When apps are invoked, Parsl intercepts the call and returns a *future* in lieu of a result. Developers can call Apps like any other Python functions and link together Apps into sophisticated workflows via standard Python code. Parsl establishes a DAG of dependencies between Apps (based on exchange of data) and sends apps for execution only when dependencies are resolved.

Parsl implements an extensible runtime model based on Python's *concurrent.futures.Executor* [9] interface as a standard way of executing tasks and a new *provider* abstraction for managing underlying compute resources. The executor abstraction has proven to be a flexible interface for integration: Parsl includes three Parsl-specific executors (HTEX, EXEX, LLEX), a standard Python ThreadPoolExecutor, and integrations with external task execution systems such as WorkQueue [10], IPyParallel, Balsam, RADICAL-Pilot, Flux, Swift/T, and funcX [11]. Parsl's provider abstraction offers a Python interface to job schedulers and cloud providers and currently supports more than one dozen providers, including Slurm, PBS, LSF, AWS, and Kubernetes.

RADICAL Cybertools (RCT) are capabilities developed in Python to support the execution of heterogeneous workflows and workloads on HPC infrastructures. RCT provides a workflow engine specialized for the execution of ensembles (Ensemble Toolkit (EnTK)), a runtime system (RADICAL-Pilot (RP)), and an interface to batch-systems via RADICAL-

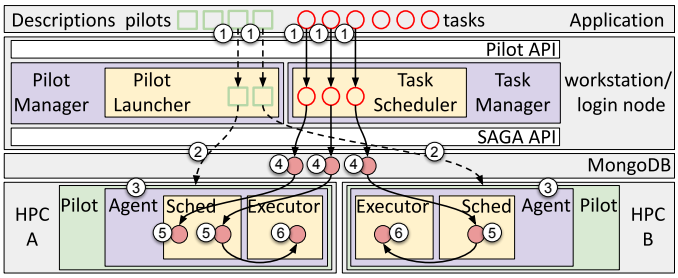


Fig. 2. RADICAL-Pilot architecture and execution model

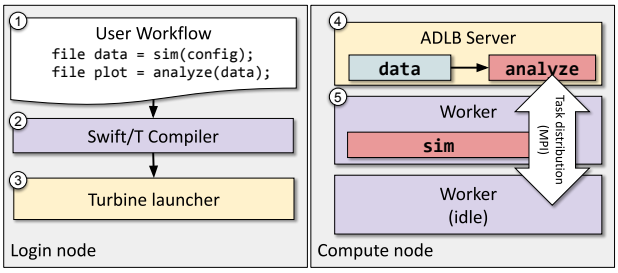


Fig. 3. Swift/T architecture and data dependency handling

SAGA (RS). Here we focus on RP [12], which is a pilot system designed to address research challenges related to efficiency, effectiveness, scalability, and both workload and resource heterogeneity. RP enables the execution of one or more workloads comprised of heterogeneous tasks on one or more HPC platforms. RP offers: (1) concurrent execution of tasks with five types of heterogeneity; (2) concurrent execution of multiple workloads on a single pilot, across multiple pilots and across multiple HPC platforms; (3) support of all major HPC batch systems to acquire and manage computing resources; (4) support of fifteen methods to launch tasks; and (5) integration with third-party workflow and runtime systems. The five types of task heterogeneity supported by RP are: (1) type of task (executable, function or method); (2) parallelism (scalar, MPI, OpenMP, or multi-process/thread); (3) compute support (CPU and GPU); (4) size (1 hardware thread to 8000 compute nodes); and duration (zero seconds to 48 hours).

RP offers an API to describe both pilots and tasks, alongside classes and methods to manage acquisition of resources, scheduling of tasks on those resources, and the staging of input and output files. Architecturally, RP is a distributed system with four modules: PilotManager, TaskManager, Agent and DB (Fig. 2, purple boxes). Modules can execute locally or remotely, communicating and coordinating over TCP/IP. PilotManager, TaskManager and Agent have multiple components where some are used only in specific deployment scenarios, depending on both workload requirements and resource capabilities. Some components can be instantiated concurrently to enable RP to manage multiple pilots, tasks and HPC resources simultaneously. This allows to scale throughput and enables component-level fault tolerance. Components are coordinated via a dedicated communication mesh implemented with ZeroMQ, which improves overall scalability of the system and lowers component complexity.

Swift/T [13] is an MPI-oriented workflow language and runtime system. Swift/T is designed to enable the execution of very large numbers of very small tasks across an MPI-enabled computing system. The tasks could be as simple as short calls to libraries implemented in compiled code, wrapped in scripting languages like Python or R, or packaged as external executables. These tasks can themselves be parallel MPI jobs launched through various mechanisms. Swift/T has been used to run ensemble applications on the largest available petascale supercomputers, such as COVID-19 population model calibration runs on Theta [14] and deep learning workflows on Summit [15].

Swift/T consists of two components. Its lower-level Turbine runtime [16] provides the MPI-level job launch, standard library, configuration features, and optional link-time integration with external scripting languages (Python, R, Tcl, JVM, Julia). Turbine also wraps around the previously developed ADLB [17] load balancer, a pure MPI library that performs work-stealing and task distribution. The higher-level STC compiler is an optimizing compiler that translates the functional Swift language constructs into a format for parallel execution and dataflow-controlled progress. As shown in Fig. 3, users use the Swift language to develop a workflow program (1) with implicitly concurrent semantics. The Swift/T compiler (STC) (2) [18] translates that into a format for execution by the Turbine runtime (3), which launches the program in parallel using the MPI implementation, possibly using a system scheduler. At run time, the ADLB Server (4) distributes tasks that are ready to run (`sim`) to workers (5), while tasks that are blocked for data (`analyze`) are held until their data dependencies are resolved.

B. Common APIs

PSI/J is a portability layer across different HPC workload managers allowing workflow developers and users to create portable workflows with a standard API. Our survey highlighted the challenge of porting workflows as one of the most critical needs for workflow developers and users alike. We further observe that most modern HPC workflows each implement their own portability layer, with varying degrees of testing, generality, and performance. PSI/J aims to focus effort by pooling into a single layer the collective knowledge obtained from these disparate efforts. PSI/J is composed of both a language-agnostic community-defined specification [19] and the specification's language-specific implementations. The high-level goals of the specification are to be lightweight, user-space, minimally prescriptive, scalable via asynchronous operations, general, and extensible to different systems, schedulers, implementations.

The PSI/J specification is broken into three layers that each build on one another. The first layer forms the base of the specification and focuses on supporting the launching and monitoring of "local" jobs (i.e., the client is running on the same system or cluster as the jobs). The second layer builds on the

first to provide support for "remote" jobs (i.e., the client must connect over the network, potentially with authentication, to launch and monitor jobs), managing multiple clusters simultaneously, and allowing file staging. The third layer builds on the previous two to support efficient execution of small jobs through the "nested", "pilot job", or "jobs-in-jobs" paradigm.

As a community, we have focused initially on implementing a PSI/J Python library [20], but implementations in other languages are encouraged. The Python library contains the set of core classes defined by the specification, as well as abstract base classes (ABCs) for components that implement functionality specific to different clusters and batch systems. The implementation currently supports `local` for running on a system without a resource manager (e.g., a user's laptop) as well as `rp` and `flux` for launching jobs under RADICAL-Pilot and Flux, respectively. Executor implementations for SAGA [21] and Slurm [22] are also under development.

C. Exaworks Software Development Kit [23]

The ExaWorks SDK aims to make workflow technologies easier to deploy, build upon in diverse applications, leverage multiple workflow systems for the same application, and interoperate with external systems. This will democratize access to increasingly hardened, scalable, and portable workflow technologies, components, and solutions to typical problems.

The SDK is implemented via a community-based approach. We follow an open community-based design process in which all artifacts are tracked using an open process on GitHub. We have defined community policies for inclusion of technologies in the SDK [24] to ensure minimum standard software quality practices for reliable deployment and use, modeled on E4S. We will work with workflows developers to integrate technologies that meet these policies. For example, we require that technologies have Docker containers and Spack packages for deployment. These packaging solutions allow for portable and reproducible deployment and testing techniques, which vary considerably among workflow tools. We have defined a Fork+Pull model-based GitHub development workflow and and GitHub Actions-based continuous integration testing and deployment for SDK components.

Importantly, we aim for the SDK to facilitate progressively advanced levels of interoperation among the tools. We define three interoperability levels as follows:

- **Level 0: Technologies can be packaged together:** A basic container or other deployment system can support technologies in the same environment.
- **Level 1: Technologies can interoperate:** A single workflow solution can use features from two or more systems which use tool-specific interfaces.
- **Level 2: Sustainable interoperability:** Users can perform deep customization of workflow system behavior, choosing from and composing tools that interoperate through the common APIs.

The next steps for the SDK are to develop and harden deeper integration of our technologies, to extend our CI/CD pipeline to Exascale Computing Project (ECP) systems, and to integrate othre community workflows tools. We expect that a richer CI/CD pipeline and more examples of deeper integration examples will significantly facilitate the rapid integration of a wider range of community workflows libraries and tools into the SDK. Thus far, we have focused on small-scale interoperation, but we plan to extend our solutions to ensure scalable integration of the components on pre-exascale and early access exascale systems for immediate readiness as exascale systems become available. We also plan to apply our SDK solutions to ECP and other exascale-relevant workflows including the ECP ExaAM workflow.

IV. Integration Examples

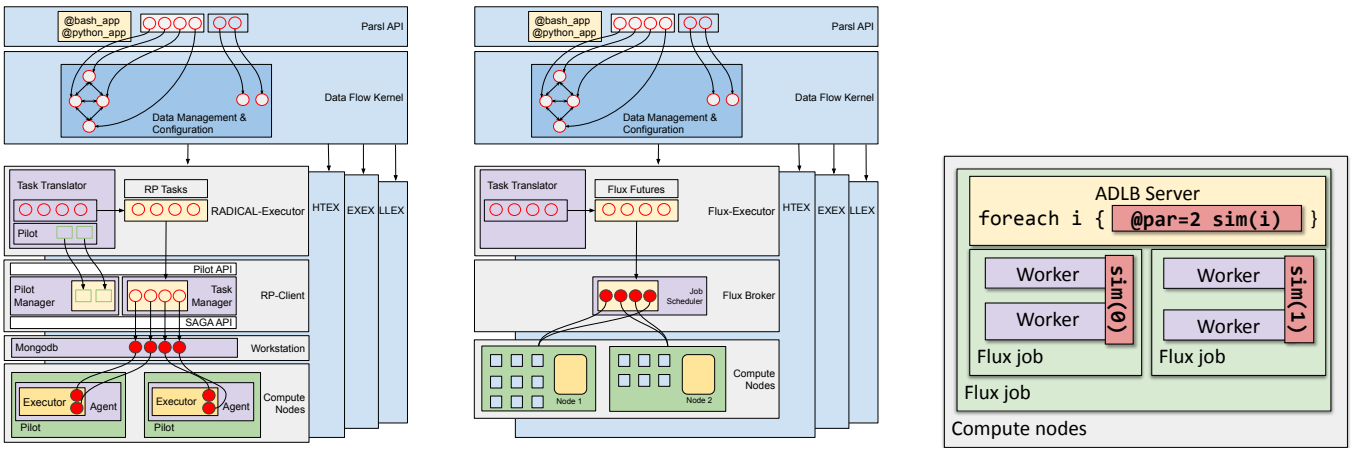
The ultimate goal of the ExaWorks SDK is to enable various components to be adopted and combined to meet use cases. Towards this goal we have prototyped integrations between SDK tools to validate the feasibility of this approach and to enhance these tools with new capabilities.

Parsl + Flux: Parsl's standard executors are not designed to schedule tasks based on resource requirements. To provide this capability we integrated Parsl and Flux such that Parsl can leverage Flux's scheduling features and support applications with varying resource requirements. To do so, we implemented Parsl's standard executor interface in Flux as shown in Fig. 4b.

Flux + RADICAL-Pilot: We extended RADICAL-Pilot (RP) to support Flux as an alternative backend system to place, launch, and manage tasks across allocated resources. The delegation of these costly operations to Flux helps to reduce RP runtime overheads. RP can increase the overall task throughput by launching multiple Flux instances within the same job allocation and using them concurrently [25]. RP starts with bootstrapping its components in the job allocation, then it launches Flux instances and schedules tasks on them for execution. Flux schedules, places and launches tasks on compute nodes via its daemons. RP Executor tracks task completion, and communicates this information to RP Scheduler, based upon which RP Scheduler passes more tasks to Flux for execution.

Parsl + RADICAL-Pilot: We integrated RP and Parsl, as shown in Fig. 4a, to provide a new scalable runtime in Parsl that is capable of managing and executing heterogeneous tasks efficiently. We designed this integration based on the structural adapter pattern [26]. The adapter pattern allows Parsl and RP to communicate seamlessly using efficient object conversion at execution time. Our performance characterization of RP-Parsl [27] showed that the overheads of RP-Parsl are small and invariant to the number of tasks, and number/type of resources.

Swift/T + Flux: We have designed an integration between Swift/T and Flux. Previously, we have developed techniques for managing large numbers of small to medium-sized MPI jobs: `Comm_create_group()` [28] and `Comm_launch()` [29], and we recognize that these features could be extended by tapping into Flux's abilities to handle a workload consisting of hierarchical or nested parallelism. For example, Swift/T could be used to specify the outermost parallel program, and parallel simulations could run inside



(a) (b) (c)

Fig. 4. Integration architectures: (a) Parsl + RADICAL, (b) Parsl + Flux and (c) Swift/T + Flux

it. More complex structures are also possible. This model is depicted in Fig. 4c where Swift/T runs inside a Flux instance.

V. Use Cases and Applications

We outline exemplar use of ExaWorks technologies in ECP applications and in extreme-scale COVID-19 research.

A. Representative Applications

We are working with several ECP application teams to understand requirements and apply ExaWorks technologies.

Cancer Distributed Learning Environment (CANDLE) is a deep learning (DL)-oriented cancer application suite for exascale. It consists of three key software products: Benchmarks, Libraries, and Supervisor [30]. The Benchmarks are a collection of relatively small, self-contained cancer applications, such as predicting the presence of a tumor in an RNA expression sample. The Libraries are a collection of tools to support DL, including configuration, I/O, checkpoint/restart, and analysis methods. Supervisor is the workflow component of CANDLE and is built on Swift/T. Supervisor workflows include flat bag-of-tasks cases, hyperparameter optimization (HPO), and more advanced data-oriented investigations [15]. The workflows allow the Benchmarks and Library features to be easily plugged in and run on a wide range of pre-exascale resources. For example, a new HPO idea could easily be applied to CANDLE-compliant Benchmarks for evaluation, and new CANDLE-compliant Benchmarks (or other DL models) can easily be developed and run at scale.

Colmena [2] is an open-source Python framework for ML-steering of simulation campaigns at scale. Colmena allows developers to define multi-fidelity simulations—either computational or ML surrogates—for determining properties. Colmena further allows users to define the control logic used to select which which simulation and ML tasks to execute when, as well as implementations of those tasks. Colmena allows for different steering functions to be defined and used to orchestrate the campaign, typically these functions are themselves ML algorithms that are repeatedly trained and applied. Colmena manages the complexity of task dispatch, results collation, ML model invocation, and ML model (re)training, using Parsl to execute tasks on HPC systems. Colmena has been used to drive electroltye design campaigns spanning an enormous design space and using thousands of nodes on the Theta supercomputer. The ML-guided steering approach is able to accelerate discovery of molecules by several orders of magnitude when compared with unguided searches.

Exascale Additive Manufacturing (ExaAM) is building a complex workflow to simulate a laser melt-pool additive manufacturing build process. The workflow is composed of several application codes, each capturing a different scale or physics of the problem. The initial ExaAM challenge problem is comprised of five phases: a full-build continuum model to set initial conditions, a hi-fidelity melt pool model that captures powder bed fusion, a detailed microstructure-scale solidification model, a polycrystal property model, and finally a continuum model that leverages the AM process-aware material model from the detailed fine-scale simulations. Each stage feeds information to the next stage, and iteration can occur across stages. Some stages leverage CPU's and others GPU's. Several stages are themselves small workflows typically leveraging the *ensemble* motif. A nuance of the model is that it follows the additive process, which means that it is possible to parallelize computations over space and time by dividing the build process into layers and then breaking the laser path into segments for each layer. Thus, the ExaAM workflow requires integration of multiple physics applications and when executed at full scale will require complex orchestration of many sub-workflows. Initial integration of ExaWorks SDK technologies has shown that improvements in throughput are possible with minor changes to existing scripts. The ExaAM and ExaWorks teams have identified additional stages in the workflow that could benefit by incrementally integrating ExaWorks SDK capabilities to improve portability and performance.

B. Gordon Bell

The winner and two of three finalists of the SC20 Gordon Bell Special Award for COVID-19 competition leveraged ExaWorks technologies. We believe that this is a demonstration of the effectiveness of the use of 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, and three of them used ExaWorks technologies. Each team developed their own tailored workflow solutions, leveraging ExaWorks technologies at key points, to enable scalability while reducing the developer time needed to support the scale and magnitude of these research efforts."

For example, the winner of the award 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 full SARS-CoV-2 virion. AI helped identify interesting conformational changes that were explored further in detail to understand the important molecular changes that occur due to the "jiggling and wiggling of atoms."

Flux was used by another finalist doing drug design to provide the scalable backbone of Livermore's Rapid COVID-19 Small Molecule Drug Design workflow. The use of Flux provided an unprecedented level of composability of workflow systems in such a way that highly complex campaigns such as drug design are easily architected in a timely fashion.

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 batteries of ML-accelerated optimization tasks. Integrating the complete CityCOVID and ML epidemiological modeling platform was aided by multiple Swift/T design features. The simulation itself 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 invoked the simulator repeatedly through library interfaces.

Another challenge was generating and coordinating the large number of single-node optimization tasks, each of which used vendor-optimized multithreaded math kernels. These single-node tasks were calls to a range of R libraries, dispatched via a custom R parallel backend. This extended the notion of workflow composability, a key 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.

VI. Building a Workflows Community

In collaboration with the NSF-funded WorkflowsRI project, we are hosting a series of workflows community summits that aim to bring the diverse workflows community together.

The first summit brought together 48 international participants representing many WMSs, with the goal to identify crucial challenges in the workflows community. The summit considered six broad themes: FAIR workflows, training and education, AI workflows, exascale challenges, APIs and interoperability, and developing workflow community.

The second summit [31] focused on technical approaches for realizing many of the challenges identified in the first summit. It included 75 workflows developers and focused on three core topics: defining common workflow patterns and benchmarks, identifying paths toward interoperability of workflow systems, and improving workflow systems' interface with legacy and emerging HPC software and hardware.

VII. A Vision for the Future

Workflow system software will become necessary components of HPC software stacks. Including workflow requirements in the procurement/requirements process for future HPC platforms and Cloud environments will aid efforts to establish community standardization around workflow APIs. This will require user and facility community engagement around defining common APIs, and eventually, widely used and shared implementations. Encouraging adoption of these common APIs in the user community while advocating for workflow requirements to enter directly into the procurement processes at facilities, will lay the foundation for workflow developers to build, maintain, and support their workflow technologies in partnership with the facilities and private sector. This will enhance the sustainability of workflow system software. Of course, for this adoption and partnering to occur, common workflow APIs and reference implementations must be widely ported and extensively tested to meet user expectations.

PSI/J represents an initial effort towards this goal, in that it is scoped to achieve both adoption by bespoke workflow developers (i.e., small teams of domain scientists), inclusion in workflow tools (starting with the ExaWorks SDK), and be tested widely across many facilities and cloud providers. The focused scope of PSI/J may allow for it to be included as a requirement in future procurements, with the possibility of it becoming a standard API provided for both users and workflow system developers.

Partnering with industry, including cloud vendors, is a key aspect of a plan to sustain a workflows SDK beyond its current ECP funding. Cloud vendors are increasingly providing workflow capabilities as well as HPC capabilities. The commercial ModSim community involved in engineering and product development, are rapidly moving to leverage cloud capabilities

in their software platforms. It is therefore important that the HPC community engage with both HPC system integrators, as well as cloud vendors, to socialize and propagate performant workflow technologies.

We will release a first version of the SDK in 2021 and then periodically release new versions as capabilities are added. Subsequent releases will include increasing use of ECP continuous integration capabilities and deployments on additional facilities. PSI/J development and releases will continue as more backends are added and it is deployed and tested at data centers. We will be co-organizing community workshops and invite the wider community to participate. Finally, all ExaWorks development activities will continue to be hosted on GitHub and are open for community participation.

We have made concrete progress at creating an SDK of workflow technologies focused on exascale workflow requirements. Our focus on establishing a rigorous continuous integration and deployment workflow, as well as our engagement with DOE facilities, is a key part of our vision to create a community curated workflows SDK. Our vision is lofty; our mandate is lucent: instantiate and grow ExaWorks into a community-owned and guided body that can contribute `de facto` standards for HPC and Cloud workflows, host reference implementations of common APIs, and curate an SDK of industrial strength components focused on running scalably on the most powerful HPC platforms available. Succeeding at this vision will depend as much on technical capabilities as on engagement with stakeholders. We invite the workflows community to participate and collaborate with us as we work to make this vision a reality.

Acknowledgements 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 Natonal Laboratory under Contract DE-AC02-06CH11357, and Brookhaven National Laboratory under Contract DESC0012704. We acknowledge the guidance of the ExaWorks Advisory Committee: Debbie Bard (LBNL/NERSC), Ian Foster (ANL/Chicago), Jack Wells (NVIDIA), Mallikarjun Shankar (OLCF/ORNL), Bill Allcock (ALCF), Daniel S. Katz (UIUC), and former members Robert Clay (formerly of SNL) and Justin Whitt (ORNL). We thank Rafael Ferreira da Silva and Henri Casanova (Workflows-RI project).

REFERENCES

[1] P. Messina, "The Exascale Computing Project," *Computing in Science Engineering*, vol. 19, no. 3, pp. 63–67, 2017.

[2] L. Ward, G. Sivaraman, G. Pauloski *et al.*, "Colmena: Scalable machine-learning-based steering of ensemble simulations for high performance computing," in *Workshop on Machine Learning in HPC Environments*, 2021.

[3] H. Bhatia, F. D. Natale, J. Y. Moon *et al.*, "Generalizable coordination of large multiscale workflow: Challenges and learnings at scale," in *to Appear in the Proceedings of Supercomputing '19: The International Conference for High Performance Computing*, ser. SC '21.

[4] "The Extreme-scale Scientific Software Stack (E4S) Web Site." [Online]. Available: https://e4s-project.github.io

[5] "ExaWorks Workflow Survey," https://exaworks.org/documents/ExaWorksSurveyReport.pdf.

[6] M. Salim, T. Uram, J. T. Childers *et al.*, "Toward real-time analysis of experimental science workloads on geographically distributed supercomputers," 2021.

[7] Flux Framework Community, "Flux Framework: A flexible framework for resource management customized for your HPC site," http://flux-framework.org, Retrieved June 20, 2021.

[8] Y. Babuji, A. Woodard, Z. Li *et al.*, "Parsl: Pervasive parallel programming in python," in *28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2019.

[9] "concurrent.futures.Executor," https://docs.python.org/3/library/concurrent.futures.html#executor-objects.

[10] P. Bui, D. Rajan, B. Abdul-Wahid *et al.*, "Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications," in *Workshop on Python for High Performance and Scientific Computing (PyHPC) at the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing)* , 2011.

[11] R. Chard, Y. Babuji, Z. Li *et al.*, "funcX: A federated function serving fabric for science," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, Jun 2020, pp. 65—76.

[12] A. Merzky, M. Turilli, M. Titov *et al.*, "Design and performance characterization of radical-pilot on leadership-class platforms," *IEEE Transactions on Parallel & Distributed Systems*, 2021. [Online]. Available: 10.1109/TPDS.2021.3105994

[13] J. M. Wozniak, T. G. Armstrong, M. Wilde *et al.*, "Swift/T: Scalable data flow programming for distributed-memory task-parallel applications," in *Proc. CCGrid*, 2013.

[14] J. Ozik, J. M. Wozniak, N. Collier *et al.*, "A population data-driven workflow for COVID-19 modeling and learning," *International Journal of High Performance Computing Applications*, vol. 35, no. 5, 2021.

[15] J. M. Wozniak, H. Yoo, J. Mohd-Yusof *et al.*, "High-bypass learning: Automated detection of tumor cells that significantly impact drug response," in *Proc. Machine Learning in High Performance Computing Environments (MLHPC) @ SC*, 2020.

[16] J. M. Wozniak, T. G. Armstrong, K. Maheshwari *et al.*, "Turbine: A distributed-memory dataflow engine for high performance many-task applications," *Fundamenta Informaticae*, vol. 28, no. 3, 2013.

[17] E. L. Lusk, S. C. Pieper, and R. M. Butler, "More scalability, less pain: A simple programming model and its implementation for extreme computing," *SciDAC Review*, vol. 17, 2010.

[18] T. G. Armstrong, J. M. Wozniak, M. Wilde, and I. T. Foster, "Compiler techniques for massively scalable implicit task parallelism," in *Proc. SC*, 2014.

[19] "Job API Specification Github Project," https://github.com/ExaWorks/job-api-spec.

[20] "J/PSI Python Implementation Github Project," https://github.com/ExaWorks/psi-j-python.

[21] The SAGA Project, RADICAL CyberTools, http://radical-cybertools.github.io/saga-python/index.html, accessed: 2015-11-5.

[22] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

[23] "ExaWorks GitHub Site." [Online]. Available: https://github.com/ExaWorks/SDK

[24] "ExaWorks SDK Community Policies," https://github.com/ExaWorks/SDK/blob/master/POLICIES.md.

[25] H. Lee, A. Merzky, L. Tan *et al.*, "Scalable hpc and ai infrastructure for covid-19 therapeutics," *Platform for Advanced Scientific Computing Conference (PASC '21), July 5–9, 2021, Geneva, Switzerland. ACM, New York, NY, USA*, 2021, https://doi.org/10.1145/3468267.3470573.

[26] K. Ayeva and S. Kasampalis, *Mastering Python Design Patterns: A Guide to Creating Smart, Efficient, and Reusable Software, 2nd Edition*, 2nd ed. Packt Publishing, 2018.

[27] A. Al-Saadi, A. Merzky, K. Chard *et al.*, "Radical-pilot and parsl: Executing heterogeneous workflows on HPC platforms," *CoRR*, vol. abs/2105.13185, 2021.

[28] J. M. Wozniak, T. Peterka, T. G. Armstrong *et al.*, "Dataflow coordination of data-parallel tasks via MPI 3.0," in *Proc. EuroMPI*, 2013.

[29] J. M. Wozniak, M. Dorier, R. Ross *et al.*, "MPI jobs within MPI jobs: A practical way of enabling task-level fault-tolerance in HPC workflows," vol. 101, 2019.

[30] J. M. Wozniak, R. Jain, P. Balaprakash *et al.*, "CANDLE/Supervisor: A workflow framework for machine learning applied to cancer research," *BMC Bioinformatics*, vol. 19, no. 18, p. 491, 2018.

[31] R. Ferreira da Silva, H. Casanova, K. Chard *et al.*, "Workflows Community Summit: Advancing the State- of-the-art of Scientific Workflows Management Systems Research and Development," Jun. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4915801