Future Generation Computer Systems 161 (2024) 502–513

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Review Article

Extreme-scale workflows: A perspective from the JLESC international community

Orcun Yildiz [a,*], Amal Gueroudji [a], Julien Bigot [b], Bruno Raffin [c], Rosa M. Badia [d], Tom Peterka [a]

[a] Argonne National Laboratory, Lemont, IL, USA
[b] Univ. Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, Gif-sur-Yvette, France
[c] Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, Grenoble, France
[d] Barcelona Supercomputing Center, Barcelona, Spain

ARTICLE INFO

Keywords:
HPC
Extreme-scale workflows
In situ
Data model
Data transport

ABSTRACT

The Joint Laboratory for Extreme-Scale Computing (JLESC) focuses on software challenges in high-performance computing systems to meet the needs of today's science campaigns, which often require large resources, consist of multiple tasks, and generate vast amounts of data. In this context, extreme-scale workflows have been the key factor in enabling scientific discoveries by helping scientists automate the dependencies and data exchanges between workflow tasks, instead of managing those manually. In this paper we present representative extreme-scale workflows and feature workflow systems developed by JLESC participating institutions. We present lessons learned while developing these tools, alongside with the open challenges and future research directions in the field of extreme-scale workflows.

1. Introduction

The Joint Laboratory for Extreme-Scale Computing (JLESC) is an international consortium gathering several of the most prominent high-performance computing (HPC) research institutions in the world [1]. JLESC was established in 2009 with the goal of addressing the most critical needs of science by promoting collaboration among its member organizations. International collaborations such as JLESC are key to enabling high-impact research by exchanging ideas, software, and resources among participating institutions. In particular, JLESC has driven the development of extreme-scale workflows through its framework to identify and exchange research directions in the field of extreme-scale computing.

Extreme-scale workflows have been developed over the years by the HPC community to help scientists orchestrate scientific workflows on HPC systems, in the face of the growing complexity of today's scientific computing together with the ever-increasing scale of HPC machines. Workflows allow scientists to describe the collection of tasks together with their dependencies and data requirements, thereby relieving users of the burden of manually managing the scheduling and data exchange of multiple workflow tasks and potentially accelerating the pace of scientific discoveries.

In this paper we feature extreme-scale workflow systems developed by JLESC partners. We characterize each system and discuss system similarities and differences. We share lessons learned while developing these systems with respect to productivity, performance, development, design trade-offs, and collaboration aspects. We also provide an outlook to the future by discussing the limitations of today's extreme-scale workflows and important research directions. We hope that the insights and lessons learned from our tools will enable a better understanding of today's extreme-scale workflows and motivate further research addressing the open challenges.

The remainder of this paper is organized as follows. Section 2 presents representative workflows in the HPC community and provides background on workflow systems, data models, and data communication mechanisms. Section 3 characterizes the featured workflow systems developed by JLESC partners in terms of workflow description, execution and data models, data transport, and applications. Section 4 presents lessons learned while developing these systems. In this section we also discuss the role of international collaboration in the development of extreme-scale workflows. We conclude in Section 5 with a discussion of open challenges and a look toward the future.

2. Background and related work

We begin with a brief background on workflow systems and present related work from a variety of teams representing the state of the art. In this background section we do not aim to provide an exhaustive survey of workflow systems. Instead, we focus on presenting popular workflow

* Corresponding author.
E-mail address: oyildiz@anl.gov (O. Yildiz).

https://doi.org/10.1016/j.future.2024.07.041
Received 20 October 2023; Received in revised form 16 May 2024; Accepted 20 July 2024
Available online 23 July 2024

systems commonly used by scientific communities that are relevant to systems we feature in Section 3. We conclude this section by reviewing related work in data models and data transfer mechanisms commonly used by workflow systems.

2.1. Workflow systems

Scientific computing typically makes use of complex scientific workflows, where a scientific workflow comprises a series of tasks that cooperate in terms of scheduling and communication as part of a larger scientific campaign. Workflow systems manage the composition of these interrelated tasks and help scientists automate the dependencies and data exchanges between tasks, instead of managing those manually. Workflow systems usually model the workflows as a directed acyclic graph (DAG), where the nodes of the graph represent workflow tasks and the edges represent the dependencies between the tasks. The two main types of workflows we consider are in situ and distributed workflows.

2.1.1. In situ workflows

With the ever-increasing gap between I/O and computation capabilities of HPC systems, in situ workflows recently have gained traction in the HPC community. In situ workflows enable performing data analysis at simulation time, where multiple workflow tasks run concurrently within a single HPC system and data are exchanged among tasks through the memory and interconnect of the system instead of physical storage. Avoiding the storage bottleneck minimizes the I/O time, reducing the time to solution. The following examples of in situ workflow systems represent the state of the art in the community.

- Adaptable I/O System, ADIOS [2], is middleware originally designed as a flexible I/O library for scientific codes, later extended to couple simulations with analysis and visualization applications through its I/O interface. ADIOS provides support for different data models and data communication mechanisms in a transparent way to the user through its runtime configuration file.
- Catalyst [3] builds on the ParaView [4] visualization tool and extends its API to process simulation data in situ. In particular, Catalyst enables visualization tasks to access the filters available in ParaView while a simulation is running. With Catalyst, visualization tasks run synchronously with the simulation in the same address space and communicate over shared memory.
- Damaris [5] is one of the first examples of software resulting from JLESC collaboration. It was initially developed to reduce I/O jitter in simulation workflows by dedicating cores for asynchronous I/O operations in multicore nodes. The Damaris dedicated cores have since been used for other purposes such as in situ visualization and data processing. Damaris supports several backends such as HDF5 and ParaView, which makes it usable in different workflow settings including in situ and post hoc processing. In its latest release, Damaris supports Dask as a result of ongoing JLESC collaboration.
- Decaf [6] is middleware for coupling parallel tasks in situ by creating communication channels over HPC interconnects through MPI. Messages among coupled tasks are passed in a distributed fashion, point to point from producer ranks to consumer ranks, without aggregating at the root of either the producer or the consumer. Decaf provides a Python API to define the workflow graph consisting of multiple parallel programs as workflow tasks. Decaf launches these parallel programs as a multiple program multiple data MPI application. The original authors of Decaf met through a JLESC collaboration.
- FlowVR [7] has been designed to ease development and execution of high-performance interactive applications harnessing the power of distributed compute nodes in a supercomputer, a cluster, or multiple interconnected clusters. FlowVR reuses and extends the dataflow paradigm: an application is seen as a set of possibly distributed modules exchanging data through ports. A module can be a parallel code, enabling automation of existing MPI codes. FlowVR relies on a daemon running on each node to route messages and uses TCP or MPI as transport layer and shared memory for in-node data exchanges.
- Henson [8] is a cooperative multitasking system for in situ processing. It uses a built-in scripting language for workflow description and employs shared objects and coroutines as its main abstractions. Through these abstractions, Henson allows colocated tasks on the same compute node to exchange data by simply passing pointers to each other with zero copies.
- Libsim [9] is built on top of VisIt [10] to perform in situ visualization by exposing simulation data to visualization engines. With Libsim, visualization tasks use the same resources as the simulation, and they take turns operating on the same time step of data. The communication between tasks uses shared memory.
- SENSEI [11] is an in situ workflow system designed with portability in mind, targeting visualization applications. SENSEI allows tool portability by interfacing with different in situ tools such as ADIOS, ParaView's Catalyst, or VisIt's Libsim. The users can select between those tools at runtime by specifying this information within the configuration file. Moreover, SENSEI supports several data communication mechanisms (e.g., shared memory, MPI).

2.1.2. Distributed workflows

Supporting scientific progress requires workflow systems to orchestrate large and complex experiments in largely distributed infrastructures. The number of workflow management systems is large and continuously increasing. In some cases, these systems are used mainly by a scientific community that tailors the solution to their needs. A sample of representative distributed workflows is listed below.

- PyCOMPSs [12] is a task-based programming model that enables the development of parallel applications and workflows. Based on annotating the Python methods and a small API, it is able to parallelize the execution at task level and to execute it in a distributed computing platform, such as clusters, clouds, container-managed clusters, and more recently edge-to-cloud environments. Recent extensions to the PyCOMPSs runtime include management of individual task faults [13], checkpointing [14], workflow provenance [15], and task nesting [16].
- Pegasus [17] is a robust and scalable system that automates the execution of a number of complex applications running on a variety of heterogeneous, distributed high-throughput, and high-performance computing environments. It is built on the principle of separation between the workflow description and workflow execution, providing the ability to port and adapt the workflow based on the target execution environment.
- Parsl [18] is a parallel programming library for Python. It exposes a task-based programming model in which developers annotate Python functions as Parsl apps. When invoked in a standard Python program, these apps are intercepted by Parsl and sent to the Parsl runtime for execution. Parsl is designed for scalability, with an extensible set of executors tailored to different use cases, such as low-latency, high-throughput, or extreme-scale execution. funcX [19] has been defined to provide a distributed function as a service platform, using Parsl's provider interface for resource management. Parsl supports multiple executers, such as Swift/TurbineExecutor [20].
- Galaxy [21] is a web-based platform initially designed for life sciences workflows. It offers a public service and a collaborative environment, which enables sharing of analysis tools, genomic data, tutorial demonstrations, persistent workspaces, and publication services. Through a web browser interface, Galaxy users can edit their workflows in a graphical editor, which are then executed through a Galaxy server.

- NextFlow [22] enables scalable and reproducible scientific workflows using software containers. Mostly used by the life sciences community, it allows the adaptation of pipelines written in the most common scripting languages. Workflows are described using a domain-specific language aiming at simplifying the implementation and the deployment of complex parallel workflows on clouds and clusters.
- Dask [23] is an open-source Python library for parallel computing. Dask scales Python code from multicore local machines to large distributed clusters in the cloud. Dask provides a familiar user interface by mirroring the APIs of other libraries in the PyData ecosystem including Pandas, scikit-learn, and NumPy. Dask emphasizes two aspects: support for big data collections and dynamic task scheduling.
- Autosubmit [24] is a lightweight workflow manager designed to meet climate research needs. It integrates the capabilities of an experiment manager, workflow orchestrator, and monitor in a self-contained application. The experiment manager allows for defining and configuring experiments, supported by a hierarchical database that ensures reproducibility and traceability. The orchestrator is designed to run workflows in research and operational mode by managing their dependencies and interfacing with local and remote hosts.
- RADICAL-Pilot (RP) [25] is a modular and extensible Python-based pilot system. It is a self-contained pilot system that can be used to provide a runtime system for workloads comprising multiple tasks. RP can be used standalone, as well as integrated with other application-level tools as a runtime system.
- Common Workflow Language (CWL) [26] is an open standard for describing how to run command line tools and connect them to create workflows. While CWL is not a workflow management system, we believe it is relevant here because of its significant adoption in the community. Tools and workflows described using CWL are portable across a variety of platforms that support the CWL standards. Using CWL, a user can easily scale complex data analysis and machine learning workflows from a single developer's laptop to massively parallel cluster, cloud, and high-performance computing environments.

2.2. Data models

If all the tasks in a workflow share the same data model, the data exchanged between the different tasks will likely use this common data model. However, when there are different data models in the same workflow, two possibilities exist: either selecting one of those models and transforming all the data to it or proposing a new data model and making all the tasks use it.

In traditional workflows, where data exchanges go through files, the associated data model is usually inspired by the file format. For example, workflows using HDF5 files use the HDF5 data model. Similarly, in visualization workflows using ParaView, VTK files employ the VTK data model. File-inspired data models can also be used for in situ workflows, bypassing physical file storage but maintaining the file semantics. For instance, LowFive [27] uses the HDF5 data model for in situ processing. VTK is commonly used in ParaView Catalyst [28] and VisIt Libsim [9] as well as in the more generic platform SENSEI [11].

Lightweight libraries such as Conduit [29] and PDI [30] have the advantage of describing the data model independently from the application, making them good candidates for in situ workflows. They are generic and describe the data model in YAML or JSON format. For instance, a simulation instrumented with PDI uses the same data model for checkpointing into HDF5 or netCDF formats for processing data in situ using SENSEI. ADIOS [2,31] is another example of a system providing its own data model that can be used by the supported backends for physical storage and in situ processing.

Bredala [32] is the data model associated with the Decaf [6] in situ system. It annotates the data fields and sends these annotations with the data, to protect their semantics while redistributing them.

Deisa [33], a Dask-enabled in situ processing library, is built on top of the PDI data model and adds a virtual global data structure to the PDI YAML to map local data into a spatiotemporal distribution. This description is shared with the Dask analytics to protect the semantics as well.

2.3. Data transfer mechanisms

One of the main capabilities of workflow systems is to automate data transfers between the workflow tasks. Data transfer mechanisms differ depending on the type of workflow system, and they also vary among different workflow systems of the same type. In situ workflows often communicate through shared memory or interconnect fabric, while distributed workflows communicate through files.

Shared memory can offer benefits for workflow systems when the tasks are on the same node by enabling zero-copy communication. In situ solutions such as Libsim and Catalyst adopt shared-memory communication between analysis and visualization tasks running synchronously with the simulation. Henson uses shared memory to exchange data between the colocated tasks on the same node by dynamically loading the executables of these tasks into the same address space.

When the tasks are located on separate compute nodes in the same system, workflow systems can communicate through the system interconnect, enabling fast communication by avoiding the parallel file system. This communication usually takes two forms: direct messaging and data staging. Systems such as Damaris and Decaf create a direct communication channel among the workflow tasks and use direct messaging via MPI to exchange data between the workflow tasks. On the other hand, some systems such as DataSpaces [34] and Colza [35] use a separate staging area, which is shared by the workflow tasks as a distributed memory space.

DataSpaces, Colza, and Deisa use remote procedure calls (RPCs) for communications. DataSpaces and Colza rely on Mercury [36] whereas Deisa relies on Dask's RPC implementation. These systems are contributing to the recent trend to employ alternative communication solutions rather than MPI to communicate through the interconnect. This trend is due mainly to limitations of MPI such as lack of elasticity or fault tolerance. The dynamicity of RPC-based systems provides an interesting alternative for irregular applications, rather than relying on a static and regular communication pattern of MPI.

File-based communication provides the least efficient data transfer mechanism for workflow systems, given the storage bottlenecks of today's machines. However, this communication method allows workflow systems to communicate between the tasks that can be located at several independent machines in a wide area. Some examples of workflow systems that use files for communication are Pegasus, PyCOMPSs, and Parsl.

3. Featured workflow systems

In this section we characterize extreme-scale workflow systems developed by JLESC participating institutions in terms of workflow specification, execution model, data model and transport, and applications.

3.1. Wilkins

Wilkins [37] is an in situ workflow system currently under development that enables dynamic heterogeneous task specification and execution for in situ processing.[1] Wilkins provides a data-centric API for defining the workflow graph, creates and launches tasks, and establishes communicators between the tasks.

[1] Wilkins is available at https://github.com/orcunyildiz/wilkins/.

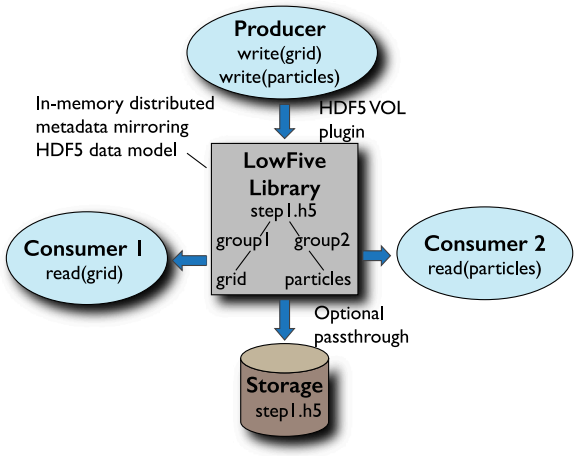

Fig. 1. Example of three tasks coupled through the LowFive in situ data transport library.

Workflow specification. Wilkins has a data-centric workflow definition, where users indicate tasks' resource and data requirements using a YAML configuration file. Based on these data requirements, Wilkins automatically creates the communication channels between the workflow tasks and generates the workflow graph as a representation of this workflow configuration file. Wilkins supports any directed-graph topology of tasks, including common patterns such as pipeline, fan-in, fan-out, ensembles of tasks, and cycles.

Execution model. The Wilkins runtime has a Python driver that reads this workflow specification upon initialization and launches the workflow. In this workflow, individual codes can be serial or parallel; they can also have different languages such as C, C++, Python, or Fortran. All the workflow functions (e.g., data transfers, data model specifications) are defined through this Python driver, with no modifications needed for the user codes. The Python driver code is provided by the Wilkins system; the user needs to supply only the constituent task codes and the workflow configuration file.

Each of the task codes can be an MPI parallel program. Although Wilkins executes the task codes in single program, multiple data (SPMD) style—in other words, the tasks are a partitioning of the global MPI_COMM_WORLD communicator, each of the tasks is written in a singular standalone fashion, using the MPI_COMM_WORLD as if it were the only user of that communicator. Wilkins manages the partitioning of the global world communicator into separate local communicators, one for each task, as well as the intercommunicators connecting tasks. This is all entirely transparent to the user task codes.

Data model. As shown in Fig. 1, Wilkins employs the data model of LowFive [27], which is a data model specification, redistribution, and communication library, implemented as an HDF5 virtual object layer (VOL) plugin. HDF5 [38] is one of the most common data models, and LowFive as a VOL plugin benefits from HDF5's rich metadata describing the data model while affording users the familiarity of HDF5.

Data transport. As its data transport layer, Wilkins uses the data redistribution components of LowFive, which supports data redistribution from M to N processes. Wilkins allows coupled tasks to communicate both in situ, using MPI message passing, and through traditional HDF5 files if needed. Users can select these different communication mechanisms in the workflow configuration file prior to workflow execution.

Applications. The LowFive library [27] has been evaluated in a high-energy physics use case, where the science campaign consisted of the Nyx [39] cosmological simulation and Reeber [40] analysis code.

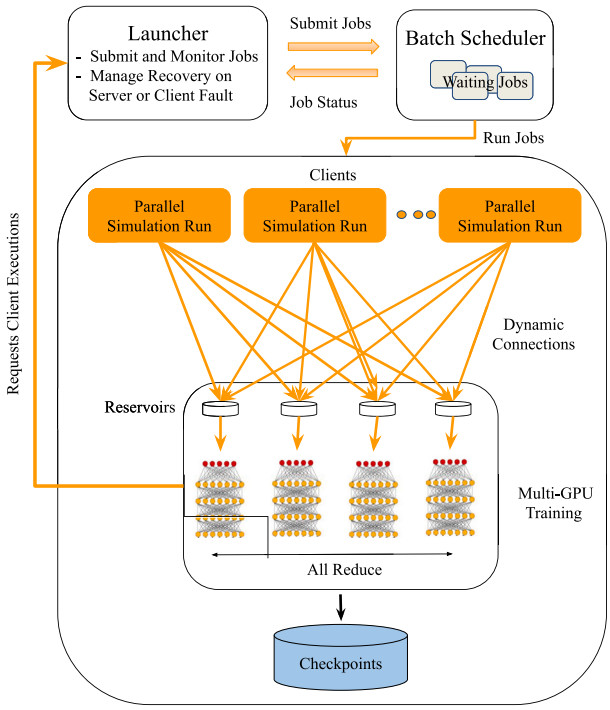

Fig. 2. Overview of Melissa architecture. The server is here set for online training a deep surrogate model with data parallelism on multiple GPUs.

LowFive enabled scalable I/O performance, while requiring no source-code modifications to the workflow tasks. Additional use cases for Wilkins in climate science and molecular dynamics are currently being developed.

3.2. Melissa

Melissa [41] is a file-avoiding, fault-tolerant, and elastic framework used to run large-scale ensembles and process them in situ. An ensemble run consists of executing the same simulation code several times but with different input parameters to get a probabilistic estimate of the simulation behavior. Ensemble runs are used for various use cases ranging from uncertainty quantification or Bayesian inference to reinforcement learning. Each instance being a parallel code often producing significant amounts of data, ensembles that can account for thousands of instances are usually very I/O intensive, especially if these results are written to disk. Adopting the in situ paradigm helps avoid this I/O bottleneck, but it requires an adapted software infrastructure to achieve performance at scale and flexibility.

Workflow specification. A Melissa workflow specification is done through a centralized file to load some predefined configurations (e.g., for using the OAR or Slurm job scheduler) and customize various aspects of the targeted run (e.g., experimental design, fault tolerance, data exchange pattern, server configuration).

Execution model. As shown in Fig. 2, Melissa's architecture relies on three main components. Melissa clients run the parallel numerical simulation code instances. The Melissa server is a parallel code in charge of processing the data received from clients. The Melissa launcher orchestrates the execution and monitor progress in tight interaction with the job scheduler. It submits client and server jobs to the job scheduler and, when required, restarts failing instances. A client can start at any time. When it starts, it dynamically connects to the server and sets a data exchange pattern adapted to the needs of the server

($N \times M$ for sensitivity analysis and data assimilation, round robin for deep surrogate training).

Data model. Melissa provides a simple data model on top of the ZeroMQ (ZMQ) messaging library [42], where messages are simple buffers. Melissa leaves to the user the responsibility to pack and unpack message buffers. Moreover, a PDI plugin for Melissa has been developed to integrate PDI-enabled simulation codes without any code changes.

Data transport. Melissa uses ZeroMQ for client/server data exchanges and MPI for the server's internal communications. ZMQ supports dynamic connections as required by Melissa's client/server architecture, but it does not provide direct support for high-performance networks. Currently, ADIOS is being integrated into Melissa as an alternative data transport and data model library to ZMQ with the goal of providing both a more advanced data model and high-performance network support.

Applications. Melissa has been used on up to tens of thousands of cores to run large-scale sensitivity analysis [43], train deep surrogate models on the fly on multiple GPUs [44], and perform data assimilation with Ensemble Kalman filters [45].

3.3. DEISA

Deisa couples MPI simulations with the Dask distributed framework for in situ processing. This system brings together in situ workflow performance with high productivity thanks to the Dask framework. Deisa maintains a good separation of concerns in the workflow by keeping the generation of data on the simulation side independent of the data handling. It uses the PDI data interface [30] to expose simulation data for external use. Fig. 3 shows the architecture of a coupled MPI application with Dask. The simulation processes (in the bottom) are instrumented with PDI and connected to *bridges* that ensure the connection and communication with Dask. Dask is represented by a single client, a scheduler, and N workers.

Workflow specification. A Deisa workflow can be seen as a hybrid task graph that includes both MPI and Dask tasks, where MPI tasks produce data and Dask tasks consume them. A task in the MPI application is all the work done by a process delimited by two communications to Dask. Since this task is computed outside of the Dask environment, it is called an external task. In order to couple the two applications, a *delivery facility* is implemented from each side. It is called a bridge from the MPI side and adaptor from the Dask side. These components ensure the connection/communications of data and metadata between MPI and Dask while maintaining their semantics.

Execution model. The Dask scheduler is launched first; it saves its connection information in a JSON file. The workers, the client, and the simulation are then launched. Upon initialization, an event is issued by each simulation process and handled by the PDI Deisa plugin, creating the bridge object associated with each MPI rank. The bridges connect to the scheduler. The bridge in rank 0 reads the YAML configuration file of PDI and sends the Deisa virtual arrays to the adaptor [46] (step 1 in Fig. 3). Those arrays describe the spatiotemporal decomposition of the data produced by the simulation. The adaptor created by the Dask client selects the data that will be needed for the analytics and returns the selections to all the bridges.

The adaptor creates as many Dask arrays as needed for analytics and returns them to the client, as any ordinary array in Dask thanks to the *external* task integration in Dask. The client submits a task graph using those arrays to the scheduler (step 2 in Fig. 3). At each timestep, the bridges get access to the data pointer and check whether their data are needed for analytics; if so, they send the data directly to a preselected worker (step 3 in Fig. 3). When the scheduler is informed that the data are in the memory of a worker, the scheduler will trigger the task transition system to schedule dependent tasks (step 4 in Fig. 3).

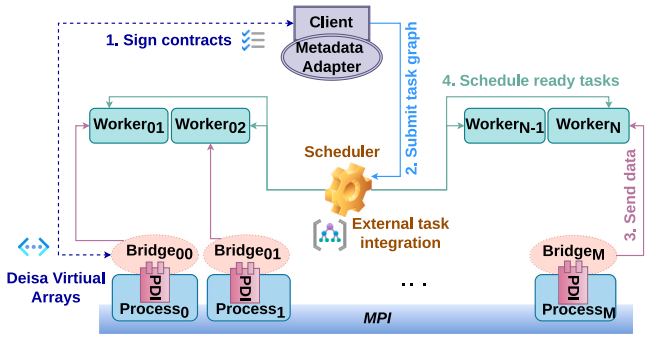

Fig. 3. Overview of Deisa architecture. The MPI application is represented by M processes; the Dask instance is represented by one client, one scheduler, and N workers. The coupling is performed through M bridges and one metadata adaptor.

Data model. Dask provides highly parallel versions of well-known libraries such as NumPy and Pandas. Dask arrays are blocked arrays that can be distributed over several nodes. Each block in a Dask array is a NumPy array or a task generating it. Dask arrays are returned by default by the adaptor. Each block in the returned Dask array is an external task, corresponding to a block of data generated by one of the MPI processes.

The semantics and position of each exchanged block of data are retrieved from the Deisa virtual arrays, which are the equivalent of the Dask arrays on the simulation side. The Deisa data model is described in two parts: the local data to each rank, including the type of the data, its subtype, and its sizes over dimensions, and the global spatiotemporal description of the data, known as Deisa virtual arrays, which includes global sizes and the position of each block in the global distribution. The local data fields are described in the PDI *data* section, which makes the description available to all loaded plugins. The global distribution description is included in the Deisa plugin specification tree because it is specific to the way data are handled by Deisa and Dask.

Data transport. The bridge sends the data block using a remote procedure call to a predefined worker. The communications between the workers are managed by Dask, transparent to the user. The only metadata that needs to be communicated to Dask are the Deisa array descriptors. The same algorithm generates the keys associated with each block in the bridges and the external tasks in the adaptor.

Applications. Deisa has been integrated into the ARK2-MHD[2] finite volume simulation code for turbulent convective dynamics on the ADASTRA[3] supercomputer. Deisa has been used to compute the power spectrum of the kinetic energy in situ using an `fft2` on 2D slices of the 3D magnetohydrodynamics to follow the evolution of the simulation, which was done by using the `fft2` from the Dask array API. Deisa makes all the Dask ecosystem usable for in situ analytics. Even ML workloads, supported by Dask-ML, are easy to build.

3.4. PyCOMPSs-Decaf

PyCOMPSs-Decaf [47] is a nested workflow, where the Decaf in situ workflow is integrated as a subworkflow in an end-to-end PyCOMPSs distributed workflow, as shown in Fig. 4. PyCOMPSs-Decaf is designed to integrate two completely different workflow programming models — task-based distributed workflows and in situ HPC workflow — with the goal of automating the full science pipeline.

[2] https://gitlab.erc-atmo.eu/erc-atmo/ark
[3] https://www.genci.fr/en/node/1149

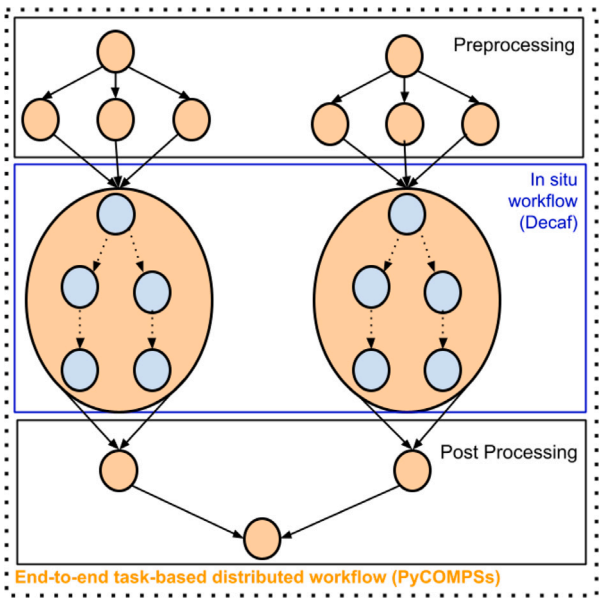

Fig. 4. Overview of the PyCOMPSs-Decaf workflow. This nested workflow composition includes two levels of hierarchy, where the orange and blue colors represent different workflow programming models.

Workflow specification. In the PyCOMPSs-Decaf workflow, PyCOMPSs manages the end-to-end workflow, where the users can program task-based workflows as Python scripts by annotating Python methods with the data and resource requirements of the tasks. Based on these requirements, PyCOMPSs automatically detects the data dependencies and generates the DAG representing the workflow.

Execution model. PyCOMPSs-Decaf executes the workflow in a master-worker fashion, where the main workflow code is executed on a master node and the task codes are executed on remote worker nodes. In PyCOMPSs-Decaf, the Decaf subworkflow appears as a normal task to PyCOMPSs, which executes it on available resources once its data dependencies are satisfied. When the Decaf subworkflow finishes, the PyCOMPSs runtime proceeds with the remaining tasks in the workflow graph that depend on the results of this Decaf execution.

Data model. PyCOMPSs-Decaf combines two different data models, where Decaf uses Bredala library and PyCOMPSs supports all object types available in Python. With Bredala, users annotate fields in a data model to indicate different data redistribution strategies. On the other hand, PyCOMPSs makes the Python objects available from one task to another by first serializing and then transferring from the source to the target node in an automatic way given that the tasks can run on distributed nodes.

Data transport. PyCOMPSs-Decaf employs two different data transport mechanisms, where Decaf exchanges data over MPI through the HPC interconnect within the in situ subworkflow, while PyCOMPSs performs data transfers among the remaining tasks in the end-to-end workflow through files.

Applications. PyCOMPSs-Decaf has been evaluated in a materials science use case, where the Decaf subworkflow consists of the LAMMPS molecular dynamics simulation coupled to a parallel in situ feature detector that identifies nucleated molecule clusters during crystallization. The PyCOMPSs workflow launches an ensemble of in situ Decaf subworkflows with different configurations, saves results from experiments with nucleation, and generates an animation by collecting frames from those ensemble members. The results show that PyCOMPSs-Decaf saves time and resources and enables better science by combining two different workflow programming models.

4. Lessons learned

4.1. Productivity and usability

Workflow systems can potentially help scientists increase their scientific productivity. However, the adoption of workflow systems still remains a challenge because scientists often have reservations when transitioning from scripting languages or manually running tasks to using a workflow system. Concerns often revolve around usability: the level of required changes to the user codes, the ease of use of the workflow interface, and whether the workflow system supports the various software ecosystems or programming models commonly found in today's heterogeneous mix of scientific computing and data analysis.

The featured extreme-scale workflow systems all use well-known languages such as YAML or Python as their workflow interface since they are easy to use. Wilkins provides a simple YAML configuration file for users to describe their workflows. This simple interface can represent different topologies such as ensembles simply by adding a single field to this configuration file. Similarly, Deisa relies on PDI for its data handling, where users can express different data requirements with a few lines of code to share data and describe them in the PDI YAML file. PyCOMPSs-Decaf allows users to define the data and resource requirements of the tasks via annotations in a Python script. Melissa uses YAML for the runtime configuration and Python for server customization.

Another barrier to entry into using workflows is the level of the required changes to the user task codes. Extensive code changes make the learning curve steeper and the barrier to adoption higher. Wilkins uses a data model based on HDF5 rather than developing a new one, since many users are already familiar with HDF5. Moreover, if the user codes already use HDF5, then no modifications are needed when executing them with Wilkins. Similarly, Deisa does not require modifications to the codes that are already instrumented with PDI. With PyCOMPSs-Decaf, the required user code changes are also minimal, where users only need to annotate their tasks with their data and resource requirements. Melissa offers two main options, a minimalist API or a PDI-plugin, to turn a simulation code into a Melissa client without any code changes.

Today's extreme-scale workflows can be heterogeneous, consisting of tasks with different programming models and languages. The workflow systems featured here also support such heterogeneity. In a JLESC collaboration integrating PyCOMPSs and Decaf, PyCOMPSs-Decaf combines different programming models such as bag-of-tasks, looping, dynamics, and parallel dataflows. Moreover, PyCOMPSs-Decaf combines Python and C++ workflow tasks for easier implementation of multilevel workflows for science applications. Deisa users have the whole Dask ecosystem available for in situ processing, with its supported distributed versions of several popular data analytics packages such as NumPy and scikit-learn. Similarly, the use of the HDF5 API in Wilkins can allow coupling HPC, big data, and AI tasks because HDF5 is commonly used by all three communities. Since Melissa servers process data in an adaptive (also often called incremental) mode, clients can run at different paces, potentially running at different levels of parallelism or on different types of resources. The server, in particular when used to train neural networks, can leverage multiple GPUs.

4.2. Performance

The performance of workflows is driven by the ever-increasing gap between computing and I/O in HPC systems, with different approaches to bypass this limitation employed by various workflow systems.

Traditional post hoc workflows write simulation data to disk before reading it back for postprocessing. When the generated data were small, such workflows functioned well; as we approach the exascale era, where numerous extreme-scale simulations generate dozens of

Table 1
Summary of the characteristics of the featured extreme-scale workflow systems.

Feature/ Workflow System	Wilkins	Melissa	Deisa	PyCOMPSs-Decaf
Workflow specification	Static (YAML)	Static (Python)	Static (Python)	Implicit (Python)
Execution model	SPMD	Client–server	MPI+X, (X=Client–server)	Master-worker
Data model	HDF5 (LowFive)	Custom (ZMQ, PDI)	Custom (PDI)	Python objects, custom (Bredala)
Data transport	MPI, files	MPI, ZMQ	RPC	MPI, files

terabytes per hour, post hoc workflows become less efficient and often infeasible.

This inefficiency is due mainly to the widening gap between computing power and I/O bandwidth. While computing power grew exponentially over the years, I/O bandwidth did not. For instance, Frontier [48], the first exascale machine, presents a gap of almost six orders of magnitude between its computing power and storage bandwidth. Such a gap makes writing all the generated data to disk infeasible due to both limited storage capacity and the required time for I/O.

The workflow community considered several approaches to reduce the impact of this I/O bottleneck on the workflows' performance in particular and scientific discovery in general.

The approaches can be divided into two main categories that are not exclusive. The first category considers bypassing the parallel file system access, by communicating either through shared memory or the HPC interconnect rather than files. In such workflows, which are known as in situ workflows, both the simulation and the analytics applications are deployed in the same platform and run concurrently. Wilkins, Deisa, and Melissa all adopt this approach.

Another approach that bypasses disk access is writing the data to burst buffers that include faster memories such as SSDs and NVRAMs instead of shared storage. Such workflows still use files for communications but take advantage of these intermediate memories to improve performance. PyCOMPSs supports an API that allows applications to store regular Python objects as persistent in new memory devices such as NVRAM or SSDs. This API is implemented both by Hecuba (based on key–value databases) and dataClay [49] (object-oriented distributed storage). This enhances data locality and optimizes the mechanism of passing parameters to tasks. Since Python processes cannot share data in memory, the exchange of parameters between tasks is normally done through serializing/deserializing them in files. With the dataClay/Hecuba libraries, however, the parameters are directly stored in the persistent memory devices, avoiding I/O overheads.

The second category considers reducing the amount of data to write. This can be done statically, by choosing only parts of the data the scientists think are interesting to study, or dynamically, by following the evolution of interesting and rare events. The static filtering of the data is easy to set up. Nevertheless, a good knowledge of the phenomena under study is required in order to avoid missing interesting events. The dynamic filtering of the data is more interesting because it smartly selects the data to keep, but it is more complicated to set up. Deisa has been used to compute the power spectrum of the kinetic energy to follow the evolution of the simulation and then trigger an up-scaling when some condition is fulfilled. Similarly, PyCOMPSs-Decaf stores only the data corresponding to nucleated molecules in the simulation rather than storing all the molecules.

4.3. Balancing performance and productivity

Rather than looking at performance and productivity in isolation, perhaps the two metrics should be considered in tandem, especially in the context of workflow systems that facilitate substituting different implementations of algorithms.

Traditionally, performance is measured in terms of floating-point operations per second (flops), but optimizing the most flops at any cost may not be the optimal way to utilize human and machine resources. For instance, to get the most efficient codes running, one is required to learn and use low-level libraries efficiently, a process that can take months or years depending on the complexity of the application.

The human productivity cost is one-time, but whether the workflow executes long enough to recoup that cost is often neglected. Therefore, we propose a holistic approach that considers both performance and productivity.

Depending on the objective, shorter run time or shorter development time can be preferable. For instance, in the scientific discovery process, agility is a key. Here, the goal is to better understand the scientific phenomena, often involving the exploration of numerous hypotheses and possibilities by scientists before obtaining new findings. Thus, only after identifying promising research directions should development resources (e.g., human, machine) be allocated to optimization and shortening run time. Also, how many times scientific code will be executed is important in deciding whether to prioritize shorter run time or shorter development time. For example, if the code will be used many times, then shorter run time becomes advantageous, thus justifying the allocation of additional development time to achieve this objective.

Metrics of human time investment are seldom evaluated in workflow research papers because they are difficult to measure. Some indirect metrics such as lines of code or cyclomatic complexity [50] have been used as proxies for productivity.

4.4. Development

When developing a new workflow system, there are often two main scenarios: either creating a single monolithic system with all the required capabilities such as data transport, workflow execution, and interface or modularizing software into different libraries and integrating them to compose a complete workflow system. The workflow systems featured here often took the latter approach during their development. For instance, Wilkins builds on top of the LowFive library, which provides data transport and data model functionalities. Similarly, Melissa relies on ZMQ as its data transport layer.

Sometimes this integration may happen at the workflow system level with the goal of combining the benefits of these different systems into a single one. For instance, PyCOMPSs-Decaf combines the advantages of in situ HPC workflows and task-based distributed workflows to automate the full science pipeline. Similarly, Deisa couples Dask's ease-of-use interface with high-performance MPI simulations.

Modularizing software comes with some challenges as well. One needs to work with other teams, since the composition process requires a good understanding of the integrated systems. For instance, while LowFive benefits from HDF5's rich data model, LowFive development required a significant amount of effort in reverse-engineering the HDF5 source code. A similar amount of effort went into coupling MPI with Dask in Deisa. Fortunately, collaborations between JLESC institutions such as the PyCOMPSs-Decaf integration reduced the required level of effort because developers for both systems were involved in the process.

Another problem can be increased code complexity when different software libraries comprise a single workflow system. This can make the software assembly and installation more complex, mixing software with different programming languages or even different programming models (e.g., task-based programming with message passing as in Deisa.) This can impede adoption of workflow systems by scientists who find it easier to work in a traditional HPC software stack. Moreover, this may also lead to an increase in the number of tuning parameters, as in the cloud ecosystem where it is common to build applications by assembling a thick stack of software components.

4.5. Similarities and differences

We summarize by discussing the similarities and differences of the extreme-scale workflows we feature along four dimensions: workflow specification, execution model, data model, and data transport layer. Table 1 shows the summary of the characteristics of the featured extreme-scale workflow systems.

Workflow specification. Workflow systems aim to provide an easy-to-use workflow description interface to the users. Through this interface, users can describe their workflows by indicating the tasks, their data and resource requirements, and the dependencies between the tasks. Wilkins uses a YAML configuration file that includes the data and resource requirements of workflow tasks. In Deisa, users define their workflows as Dask task graphs using a Python API. In Melissa, users provide a script with the simulation groups and parameter sets to the Melissa launcher for instantiating the run. Unlike these systems that use a static script for workflow description, PyCOMPSs-Decaf implicitly generates the workflow graph based on the annotated data requirements of the tasks in a Python script. Having an implicit workflow description allows PyCOMPSs-Decaf to support dynamic workflows with data dependent behaviors.

Execution model. Deisa relies on Dask for its execution model as it aims to bring the benefits of the distributed task-based paradigm to MPI simulations. For instance, Dask is popular in the data analytics community with its support for parallel versions of NumPy, Pandas, and scikit-learn. Similarly, PyCOMPSs-Decaf supports different programming models such as dynamicity and bag-of-tasks thanks to its task-based execution model. Wilkins executes the workflow tasks as a single SPMD MPI application. Melissa is specialized for ensemble runs and enforces modularity and flexibility by handling clients and servers as independent codes that can dynamically connect once running, a key feature for enabling fault tolerance, load balancing, and elasticity.

Data model. Workflow systems either introduce a new data model of their own or reuse existing data models. Wilkins uses LowFive, which is based on the HDF5 data model. Similarly, Deisa relies on the PDI data model for its communication. On the other hand, PyCOMPSs-Decaf uses the Bredala library, which is an independent data model. One advantage of using an independent data model is extra flexibility and customization. For example, the users can use several different annotations through Bredala's API to indicate different data redistribution strategies. However, custom annotations require modifications to the task codes and incur a higher learning curve for using this new data model. On the other hand, user codes with existing data models such as HDF5 or PDI can be easily integrated into the workflow systems such as Wilkins and Deisa with minimal or no modifications to the user task codes. Similarly, a PDI plugin enables Melissa to integrate PDI-enabled codes as clients without code modifications.

Data transport. The featured workflow systems differ in their data transport mechanisms as they target different user groups. Melissa was first developed using the ZMQ library, which is designed to support dynamic client/server connections with features such as automatic reconnections, multithreading for overlapping data transfer with message handling, and client- and server-side buffering to absorb temporary imbalance between data production and consumption. ZMQ proved easy to deploy and reliable on various supercomputers. However, ZMQ does not efficiently support high-performance networks. An ongoing development focuses on using ADIOS as an alternative data transport layer to provide high-performance network support. Deisa targets combining MPI-based simulations and task-based paradigms. To this end, Deisa performs data transfers between MPI simulations and Dask workers using RPC. Both Deisa and Melissa use a single server with multiple parallel workers or clients, which can create a bottleneck in some scenarios. On the other hand, Wilkins performs parallel data transfers between different numbers of processes point to point (M producers to N consumers) using MPI. Having MPI as the communication framework results in efficient data transfers over HPC interconnects, but it comes with its own set of limitations such as a lack of support for failures or elasticity. PyCOMPSs-Decaf uses a hybrid data transport scheme: communicating over MPI when inside a Decaf in situ subworkflow and communicating using files when in the PyCOMPSs distributed workflow.

4.6. Collaboration

International collaborations within the JLESC framework have played a key role in the development of the featured extreme-scale workflow systems by creating a platform to exchange ideas, identify research directions, and establish a roadmap for their implementation. In particular, extreme-scale workflow systems need to be coordinated at large-scale resources, and we have access to large-scale HPC systems through JLESC. Moreover, each participating institution often has different science use cases based on the priorities of their research agenda; hence, exchanging different use cases has been another important advantage in the development of these featured workflow systems.

Besides facilitating software development, JLESC plays an important role in workforce development. JLESC provides training opportunities for junior researchers by learning from top international researchers in extreme-scale computing. Also, JLESC has an established history of supporting internships for graduate students, which has proved to be the key enabling factor for generating research results. Numerous students have been hired by partner JLESC institutions after graduation.

5. Outlook to the future

Despite the progress made as a result of JLESC collaboration, many challenges persist, seeding future research directions and collaborations.

Elasticity. During the course of a workflow execution, we can observe complex dynamic patterns in the behavior of the workflow tasks. For example, the analysis requirements of a simulation can evolve over time depending on the scientific phenomena, and such requirements may require different amounts of analysis resources or even new analysis tasks. Such patterns will be even more evident with the growing number of AI tasks that are dynamic by nature.

One key limitation of existing workflow systems is that they often use a fixed amount of resources. Resource descriptions are often indicated in a static configuration file by users before the workflow execution and remain constant until the workflow completion. Some earlier efforts address this issue by providing an API that can be embedded in the user codes, where users can trigger dynamic changes depending on the requirements of the workflow tasks [51]. On the other hand, some of the task-based workflow systems such as PyCOMPSs or Swift [52] allow users to program such dynamic changes in their task-based parallel workflows.

Another challenge is the reconfiguration of the workflow upon a dynamic change. Dynamic changes may include redistribution of resources among workflow tasks, or there may be scenarios where the workflow task graph changes after addition or deletion of some tasks. Such changes require workflow systems to reconfigure communication channels between workflow tasks without significant overhead. Workflow systems relying on MPI for high-performance communication are restricted by MPI's limited support for elasticity [53]. To overcome MPI's limitations, recent work uses a custom communication layer in ParaView that replaces MPI to enable elastic in situ visualization [35]. The rigidity of MPI is also the main reason that Melissa relies on ZMQ to provide an elastic framework for ensemble analysis.

We also note that enabling elastic workflows does not solely depend on the workflow community. The reason is that workflows are complex and interact with several different entities ranging from job

schedulers to user applications. For example, even if workflow systems would support dynamic changes, job schedulers are rigid and often do not allow resizing of the jobs (e.g., currently, Slurm does not allow increasing the number of resources allocated to the job). Workflow systems try to work around these limitations by integrating their own local scheduler to support a two-level allocation strategy: large enough resource allocations requested to the machine job scheduler and then fast workflow-specific scheduling within these allocations. This strategy, however, comes at the cost of extra code complexity and loss of efficiency and often limits the deployment of elasticity within workflows. Moreover, some user applications are not designed to be elastic. Given these external challenges, the workflow community needs to work together with various groups to enable elasticity on extreme-scale systems.

Artificial intelligence and extreme-scale workflows. Traditionally, extreme-scale systems have been used for their large computing power. A recent trend is to run AI applications on these systems as well, given that today's science campaigns often include HPC and AI tasks. For instance, AI methods can be used for a better understanding of HPC simulations, such as inverse modeling of data generated by HPC simulations. Thus, it is imperative for workflow systems to support running AI tasks together with HPC tasks.

One major challenge is that HPC and AI technologies come from different communities, and their programming models and software tools remain largely incompatible. Workflow systems need to mitigate such discrepancies when running HPC and AI tasks as part of a single workflow. One effort in this direction has been started in the EuroHPC JU eFlows4HPC project[4] aiming at a software stack for the development of workflows that combine traditional HPC with AI and data analytics. In the project, PyCOMPSs has been extended with new annotations to better support the integration of software written with different programming models [54]. Melissa handles the server and clients as different executables. The servers are written in Python, making it easy to integrate with deep learning libraries such as Pytorch and Tensorflow, as done in [44]. This integration is transparent to the simulation code.

Another problem is the lack of support for co-scheduling of heterogeneous resources on extreme-scale systems. Since AI tasks require GPU resources and other traditional HPC tasks run on CPUs, co-scheduling these resources for a single workflow job is another challenge. When the workflow requires a combination of CPU and GPU nodes, often distinct allocations of uniform resources need to be requested from the job scheduler, even when these nodes are on the same machine. Co-scheduling has become less of an issue in recent years as most supercomputers today feature heterogeneous processors in the same machine. However, this can still remain a problem because these resources are often handled by different scheduling queues. As a result, the workflow system may need to put some allocated resources on hold, waiting for others to be available, as is the case with Melissa when clients run on CPUs while the server requires GPU nodes for training [44].

Besides using workflows to run AI, there can be significant benefits for extreme-scale workflows when augmenting them with AI. We can easily envision AI being used to improve performance and efficient usage of resources in extreme-scale systems. For instance, the configuration and optimization of workflows can be improved with AI, which can provide optimal configurations instead of manual ones given by users. AI can also serve to characterize workflows and identify performance bottlenecks or predict failures. Such potential benefits should be explored by the workflow community in collaboration with AI experts.

[4] https://eflows4hpc.eu/

Adoption. The adoption of workflow systems by end users is one of the major challenges we encounter today. Our interactions with domain scientists and potential end-users often raise several questions. Why should I use a workflow system? Which system would best fit my use case? Do I need to make changes to my code to make it work? Will there be support if my code does not work or perform well? Will the workflow system work on a different platform or architecture? We believe these are all valid concerns, and here we reflect on how we can address them.

Dissemination and application engagement efforts can play a key role in facilitating the adoption of workflow systems. Since workflow developers and users often come from different teams or organizations, it may be difficult for users to know which workflow technologies exist or which workflow system best meets their application needs. There is a recent effort in gathering the workflow research community together [55], which highlights the need for standardization among existing worklow technologies. This effort also targets creating a workflow knowledge base for the benefit of workflow users. The community should continue along similar lines and spend more effort in reaching out to domain scientists and potential end-users.

The usability of the workflow systems is another important factor for adoption. As highlighted in Section 4, workflow systems should be easy to use while being able to express different requirements of users. Another common concern is the amount of required changes to user codes. Good documentation and tutorials for workflow systems together with other dissemination and training efforts for users can further facilitate the adoption of workflow systems.

The longevity of workflow software is another concern for users. Many inactive workflow systems have no support personnel. One of the main reasons is short funding cycles of research projects that pay for the initial development of such systems but not their prolonged sustainability. Consequently, users often have reservations about investing time in adopting a workflow system. This is an intrinsic problem of research software products not limited to workflow systems; however, recent promising efforts have been targeting software sustainability [56,57], which can help solve this problem.

Performance and performance portability is another critical element for adoption. Each new workflow management approach designed by computer scientists typically targets providing the best performance possible (as well as other improvements) on extreme-scale machines. This is often achieved by leveraging specific aspects of a given hardware to the best performance. In order to reach adoption, the integration of these new approaches should not require any important modifications to the user codes. This means that in addition to these innovative approaches, forward-thinking APIs able to support all of the approaches should be designed, with production-level backends offering the best approach on each type of hardware. Without such an effort, users will tend either to rely on a single well-tested approach or to implement directly in their code their own approaches matching their specific needs and hardware they use.

Workflow characterization. Today's scientific workflows often consist of several components with different characteristics, complicating their understanding when they are applied in practice to different use cases. Currently, no workflow standards or theory exists to help understand workflow performance, which can be key to designing robust and explainable workflows.

The lack of understanding is mainly caused by insufficient characterization of workflow performance with respect to its configuration, which is a large and high-dimensional input space. For example, this input space can include number of tasks, scale and performance profile of each task, and system characteristics. To tackle the characterization problem, one possible solution could be to develop mini-apps for workflows emulating today's complex scientific workflows. Such mini-apps have been successful in other domains for characterization and understanding of the applications, and the workflow community could adopt a similar concept.

Another required capability for workflow characterization is availability of profiling tools. Such profilers could help better understand workflows by providing insights on aspects such as scaling profiles of tasks, load imbalances, and hotspots with respect to resource assignments of tasks. One possible direction would be to adopt or extend existing profilers to workflows. For instance, there is preliminary work on understanding I/O profiles of workflows using the Darshan I/O characterization tool [58]. While adopting a profiler for HPC applications can have advantages, however, it may not be possible to fully repurpose such tools — designed for single jobs — to workflows.

Integrating different software for workflow composition. Developing a monolithic system that is good for all needed aspects within a single software workflow can be difficult. Workflow systems are complex and provide many different capabilities. A more practical solution is to integrate different software libraries comprising a workflow system or even integrate different workflow systems into a single one. This is the approach we have seen with the featured workflow systems, which proved to be beneficial when developing these systems.

Nevertheless, the integration process has its own challenges. As we discussed in Section 4, these challenges may include working with other groups and increased code complexity. In addition, integration can require extra effort to maintain compatibility among different software products, especially if such software is actively being developed.

In order to facilitate the integration process, one possible solution could be to standardize this process, where the workflow community defines and standardizes for integration of several software libraries or systems. For instance, there can be a common interface for workflow composition without being specific to any software.

CRediT authorship contribution statement

Orcun Yildiz: Conceptualization, Methodology, Writing – original draft. **Amal Gueroudji:** Conceptualization, Writing – original draft. **Julien Bigot:** Writing – review & editing. **Bruno Raffin:** Funding acquisition, Writing – review & editing. **Rosa M. Badia:** Funding acquisition, Writing – review & editing. **Tom Peterka:** Funding acquisition, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is a collaboration between Argonne National Laboratory, INRIA, and Barcelona Supercomputing Center within the Joint Laboratory for Extreme-Scale Computing. This research is supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC02- 06CH11357, program manager Margaret Lentz.

The BSC author acknowledges the support of the Spanish Government (PID2019-107255 GB) and by MCIN/AEI/10.13039/50110001 1033 (CEX2021-001148-S), by the Departament de Recerca i Universitats de la Generalitat de Catalunya to the Research Group MPiEDist (2021 SGR 00412). The same author acknowledges the European Commission's Horizon 2020 Framework program and the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 955558 and by MCIN/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR (PCI2021-121957), project eFlows4HPC.

The INRIA author acknowledges the European Union's Horizon research and innovation programme under grant agreement No 101144 014.

References

[1] Joint Laboratory for Extreme-Scale Computing, https://jlesc.github.io/.

[2] J.F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, C. Jin, Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS), in: Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments, 2008, pp. 15–24.

[3] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, J. Mauldin, ParaView catalyst: Enabling in situ data analysis and visualization, in: Proceedings of the First Workshop on in Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ACM, 2015, pp. 25–29.

[4] J. Ahrens, B. Geveci, C. Law, 36 ParaView: An end-user tool for large-data visualization, Vis. Handb. (2005) 717.

[5] M. Dorier, G. Antoniu, F. Cappello, M. Snir, R. Sisneros, O. Yildiz, S. Ibrahim, T. Peterka, L. Orf, Damaris: Addressing performance variability in data management for post-petascale simulations, ACM Trans. Parallel Comput. 3 (3) (2016) 15.

[6] O. Yildiz, M. Dreher, T. Peterka, Decaf: Decoupled dataflows for in situ workflows, in: In Situ Visualization for Computational Science, Springer, 2022, pp. 137–158.

[7] M. Dreher, B. Raffin, A flexible framework for asynchronous in situ and in transit analytics for scientific simulations, in: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE, 2014, pp. 277–286.

[8] D. Morozov, Z. Lukic, Master of puppets: Cooperative multitasking for in situ processing, in: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing, ACM, 2016, pp. 285–288.

[9] B. Whitlock, J.M. Favre, J.S. Meredith, Parallel in situ coupling of simulation with a fully featured visualization system, in: Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization, EGPGV '11, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2011, pp. 101–109, http://dx.doi.org/10.2312/EGPGV/EGPGV11/101-109.

[10] S. Ahern, E. Brugger, B. Whitlock, J.S. Meredith, K. Biagas, M.C. Miller, H. Childs, VisIt: Experiences with sustainable software, 2013, arXiv preprint arXiv:1309.1796.

[11] U. Ayachit, B. Whitlock, M. Wolf, B. Loring, B. Geveci, D. Lonie, E. Bethel, The SENSEI generic in situ interface, in: Proceedings of the 2nd Workshop on in Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, IEEE Press, 2016, pp. 40–44.

[12] E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R.M. Badia, J. Torres, T. Cortes, J. Labarta, PyCOMPSs: Parallel computational workflows in Python, Int. J. High Perform. Comput. Appl. 31 (1) (2017) 66–82.

[13] J. Ejarque, M. Bertran, J.Á. Cid-Fuentes, J. Conejero, R.M. Badia, Managing failures in task-based parallel workflows in distributed computing environments, in: European Conference on Parallel Processing, Springer, 2020, pp. 411–425.

[14] P. Vergés, F. Lordan, J. Ejarque, R.M. Badia, Task-level checkpointing system for task-based parallel workflows, in: European Conference on Parallel Processing, Springer, 2022, pp. 251–262.

[15] R. Sirvent, J. Conejero, F. Lordan, J. Ejarque, L. Rodríguez-Navas, J.M. Fernández, S. Capella-Gutiérrez, R.M. Badia, Automatic, efficient and scalable provenance registration for FAIR HPC workflows, in: 2022 IEEE/ACM Workshop on Workflows in Support of Large-Scale Science, WORKS, IEEE, 2022, pp. 1–9.

[16] F. Lordan, G. Puigdemunt, P. Vergés, J. Conejero, J. Ejarque, R.M. Badia, Hierarchical management of extreme-scale task-based applications, in: European Conference on Parallel Processing, Springer, 2023, pp. 111–124.

[17] E. Deelman, D. Gannon, M. Shields, I. Taylor, Workflows and e-Science: An overview of workflow system features and capabilities, Future Gener. Comput. Syst. 25 (5) (2009) 528–540.

[18] Y. Babuji, A. Woodard, Z. Li, D.S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. Wozniak, I. Foster, M. Wilde, K. Chard, Parsl: Pervasive parallel programming in Python, in: 28th ACM International Symposium on High-Performance Parallel and Distributed Computing, HPDC, 2019, http://dx.doi.org/10.1145/3307681.3325400.

[19] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, K. Chard, Funcx: A federated function serving fabric for science, in: Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing, 2020, pp. 65–76.

[20] T.G. Armstrong, J.M. Wozniak, M. Wilde, I.T. Foster, Compiler techniques for massively scalable implicit task parallelism, in: SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2014, pp. 299–310.

[21] V. Jalili, E. Afgan, Q. Gu, D. Clements, D. Blankenberg, J. Goecks, J. Taylor, A. Nekrutenko, The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2020 update, Nucleic Acids Res. 48 (W1) (2020) W395–W402.

[22] P. Di Tommaso, M. Chatzou, E.W. Floden, P.P. Barja, E. Palumbo, C. Notredame, Nextflow enables reproducible computational workflows, Nature Biotechnol. 35 (4) (2017) 316–319.

[23] M. Rocklin, Dask: Parallel computation with blocked algorithms and task scheduling, in: Proceedings of the 14th Python in Science Conference, (130–136) Citeseer, 2015.

[24] D. Manubens-Gila, J. Vegas-Regidora, M. Acostaa, C. Prodhommea, O. Mula-Vallsa, K. Serradell-Marondaa, F. Doblas-Reyes, Autosubmit: a versatile tool for managing earth system models on HPC platforms, Future Gener. Comput. Syst. (2016) submitted for publication.

[25] A. Merzky, M. Turilli, M. Maldonado, M. Santcroos, S. Jha, Using pilot systems to execute many task workloads on supercomputers, in: Job Scheduling Strategies for Parallel Processing: 22nd International Workshop, JSSPP 2018, Vancouver, BC, Canada, May 25, 2018, Revised Selected Papers 22, Springer, 2019, pp. 61–82.

[26] P. Amstutz, M.R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, J. Kern, D. Leehr, H. Ménager, et al., Common workflow language, v1. 0. specification, 2016, http://dx.doi.org/10.6084/m9.figshare.3115156.v2, Common Workflow Language working group.

[27] T. Peterka, D. Morozov, A. Nigmetov, O. Yildiz, B. Nicolae, P.E. Davis, LowFive: in situ data transport for high-performance workflows, in: IPDPS'23: The 37th IEEE International Parallel and Distributed Processing Symposium, 2023.

[28] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, J. Mauldin, ParaView catalyst: Enabling in situ data analysis and visualization, in: Proceedings of the First Workshop on in Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, in: ISAV2015, Association for Computing Machinery, New York, NY, USA, 2015, pp. 25–29, http://dx.doi.org/10.1145/2828612.2828624.

[29] C. Harrison, M. Larsen, B.S. Ryujin, A. Kunen, A. Capps, J. Privitera, Conduit: A successful strategy for describing and sharing data in situ, in: 2022 IEEE/ACM International Workshop on in Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV, 2022, pp. 1–6, http://dx.doi.org/10.1109/ISAV56555.2022.00006.

[30] C. Roussel, K. Keller, M. Gaalich, L.B. Gomez, J. Bigot, PDI, an approach to decouple I/O concerns from high-performance simulation codes, 2017.

[31] D.A. Boyuka, S. Lakshminarasimham, X. Zou, Z. Gong, J. Jenkins, E.R. Schendel, N. Podhorszki, Q. Liu, S. Klasky, N.F. Samatova, Transparent in situ data transformations in adios, in: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE, 2014, pp. 256–266.

[32] M. Dreher, T. Peterka, Bredala: Semantic data redistribution for in situ applications, in: 2016 IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2016, pp. 279–288.

[33] A. Gueroudji, J. Bigot, B. Raffin, DEISA: dask-enabled in situ analytics, in: 2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics, HiPC, IEEE, 2021, pp. 11–20.

[34] C. Docan, M. Parashar, S. Klasky, Dataspaces: an interaction and coordination framework for coupled simulation workflows, Cluster Comput. 15 (2) (2012) 163–181.

[35] M. Dorier, Z. Wang, U. Ayachit, S. Snyder, R. Ross, M. Parashar, Colza: Enabling elastic in situ visualization for high-performance computing simulations, in: 2022 IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2022, pp. 538–548.

[36] J. Soumagne, D. Kimpe, J. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, R. Ross, Mercury: Enabling remote procedure call for high-performance computing, in: 2013 IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2013, pp. 1–8.

[37] O. Yildiz, D. Morozov, A. Nigmetov, B. Nicolae, T. Peterka, Wilkins: HPC in situ workflows made easy, 2024, arXiv preprint arXiv:2404.03591.

[38] M. Folk, G. Heber, Q. Koziol, E. Pourmal, D. Robinson, An overview of the HDF5 technology suite and its applications, in: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, 2011, pp. 36–47.

[39] A.S. Almgren, J.B. Bell, M.J. Lijewski, Z. Lukić, E. Van Andel, Nyx: A massively parallel AMR code for computational cosmology, Astrophys. J. 765 (1) (2013) 39.

[40] B. Friesen, A. Almgren, Z. Lukić, G. Weber, D. Morozov, V. Beckner, M. Day, In situ and in-transit analysis of cosmological simulations, Comput. Astrophys. Cosmol. 3 (1) (2016) 1–18.

[41] M. Schouler, R.A. Caulk, L. Meyer, T. Terraz, C. Conrads, S. Friedemann, A. Agarwal, J.M. Baldonado, B. omiej Pogodziński, A. Sekuła, A. Ribes, B. Raffin, Melissa: coordinating large-scale ensemble runs for deep learning and sensitivity analyses, J. Open Source Softw. 8 (86) (2023) 5291, http://dx.doi.org/10.21105/joss.05291.

[42] P. Hintjens, ZeroMQ: Messaging for Many Applications, O'Reilly Media, Inc., 2013.

[43] T. Terraz, A. Ribes, Y. Fournier, B. Iooss, B. Raffin, Melissa: Large scale in transit sensitivity analysis avoiding intermediate files, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2017, p. 61.

[44] L. Meyer, M. Schouler, R.A. Caulk, A. Ribés, B. Raffin, High throughput training of deep surrogates from large ensemble runs, in: SC 2023 - the International Conference for High Performance Computing, Networking, Storage, and Analysis, ACM, Denver, CO, United States, 2023, pp. 1–14.

[45] S. Friedemann, B. Raffin, An elastic framework for ensemble-based large-scale data assimilation, Int. J. High Perform. Comput. Appl. URL https://hal.inria.fr/hal-03017033.

[46] A. Gueroudji, J. Bigot, B. Raffin, R. Ross, Dask-extended external tasks for HPC/ML in transit workflows, in: Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis, SC-W 2023, ACM, New York, NY, USA, 2023, p. 8, http://dx.doi.org/10.1145/3624062.3624151.

[47] O. Yildiz, J. Ejarque, H. Chan, S. Sankaranarayanan, R.M. Badia, T. Peterka, Heterogeneous hierarchical workflow composition, Comput. Sci. Eng. (2019).

[48] Frontier, URL https://www.olcf.ornl.gov/frontier/.

[49] J. Martí, A. Queralt, D. Gasull, A. Barceló, J.J. Costa, T. Cortes, Dataclay: A distributed data store for effective inter-player data sharing, J. Syst. Softw. 131 (2017) 129–145.

[50] J.Á. Cid-Fuentes, P. Alvarez, R. Amela, K. Ishii, R.K. Morizawa, R.M. Badia, Efficient development of high performance data analytics in Python, Future Gener. Comput. Syst. 111 (2020) 570–581.

[51] O. Yildiz, D. Morozov, B. Nicolae, T. Peterka, Dynamic heterogeneous task specification and execution for in situ workflows, in: 2021 IEEE Workshop on Workflows in Support of Large-Scale Science, WORKS, IEEE, 2021, pp. 25–32.

[52] J.M. Wozniak, T.G. Armstrong, M. Wilde, D.S. Katz, E. Lusk, I.T. Foster, Swift/T: large-scale application composition via distributed-memory dataflow processing, in: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, IEEE, 2013, pp. 95–102.

[53] M. Dorier, O. Yildiz, T. Peterka, R. Ross, The challenges of elastic in situ analysis and visualization, in: Proceedings of the Workshop on in Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, 2019, pp. 23–28.

[54] J. Ejarque, R.M. Badia, L. Albertin, G. Aloisio, E. Baglione, Y. Becerra, S. Boschert, J.R. Berlin, A. D'Anca, D. Elia, et al., Enabling dynamic and intelligent workflows for HPC, data analytics, and AI convergence, Future Gener. Comput. Syst. 134 (2022) 414–429.

[55] R.F. da Silva, H. Casanova, K. Chard, D. Laney, D. Ahn, S. Jha, C. Goble, L. Ramakrishnan, L. Peterson, B. Enders, et al., Workflows community summit: Bringing the scientific workflows community together, 2021, arXiv preprint arXiv:2103.09181.

[56] J. Ahrens, T. Gamblin, T. Germann, X.S. Li, L.C. McInnes, K. Mohror, T. Munson, S. Shende, R. Thakur, J. Vetter, et al., Toward a post-ECP software sustainability organization (PESO).

[57] A. Dubey, P. Lead, K. Harms, R. Gerber, J. Macauley, B. Messer, COLABS: Collaboration of ORNL, LBNL, and ANL for better software.

[58] J. Lüttgau, S. Snyder, P. Carns, J.M. Wozniak, J. Kunkel, T. Ludwig, Toward understanding I/O behavior in HPC workflows, in: 2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems, PDSW-DISCS, IEEE, 2018, pp. 64–75.

Orcun Yildiz is an assistant computer scientist with the Mathematics and Computer Science Division at Argonne National Laboratory. He received his Ph.D. degree in Computer Science from Ecole Normale Superieure de Rennes, France in December 2017. His research interests include in situ workflows, HPC, Big Data, and AI convergence, and I/O management.

Amal Gueroudji is a Postdoctoral Appointee at Argonne National Laboratory. She received her Ph.D. from the University of Grenoble Alpes under the Marie Sklodowska Curie Actions (MSCA) funding at the French Atomic Energy and Alternative Energies Commission (CEA). Her research interests include programming models for distributed computing, in situ processing, and data analytics workflows, with a specific focus on HPC, AI, and Big Data convergence.

Julien Bigot is a research scientist at CEA, France, where he leads the *science of computing* team of *maison de la simulation*. He defended his Ph.D. thesis in computer science in 2010 (Inria, INSA Rennes, ENS Lyon). His main research interest is related to programming models for HPC. He first worked on component based software engineering for HPC, and then joined maison de la simulation to work closer with applications, like Gysela5D. He now works on separation of concerns between performance optimization and semantic expression; more specifically GPU programming and data handling (IO, in situ processing, etc.).

Bruno Raffin is Research Director at INRIA, France, and leader of the DataMove team. He led research on middlewares for large-scale data-flow oriented parallel applications for scientific visualization, computational steering, in situ data analytics, ensemble-based sensibility analysis, data assimilation, deep surrogate training. He also worked on parallel algorithms and cache-efficient parallel data structures (cache oblivious mesh layouts, parallel adaptive sorting), strategies for task-based programming on multi-CPU and multi-GPU machines. Bruno Raffin accounts for more than 60 international publications, 17 advised Ph.D. students.

Rosa M. Badia holds a Ph.D. from the Universitat Politècnica de Catalunya (UPC, 1994). She is the manager of the Workflows and Distributed computing group at the Barcelona Supercomputing Center (BSC) and part time lecturer at the UPC. Her research focuses on programming models for distributed computing. Recently, she has been focusing on tools for the development of workflows that combine HPC, AI and data analytics. Dr Badia has published near 200 papers in international conferences and journals on the topics of her research. She is the global PI of the EuroHPC project eFlows4HPC.

Tom Peterka is a computer scientist at Argonne National Laboratory, scientist at the University of Chicago Consortium for Advanced Science and Engineering (CASE), and fellow of the Northwestern Argonne Institute for Science and Engineering (NAISE). His research interests are large-scale parallel in situ analysis of scientific data. Recipient of the 2017 DOE early career award and five best paper awards, Peterka has published over 140 peer-reviewed articles and papers since earning his Ph.D. in computer science from the University of Illinois at Chicago in 2007.