Novel Approaches Toward Scalable Composable Workflows in Hyper-Heterogeneous Computing Environments

Jonathan Bader, James Belak, Matthew Bement, Matthew Berry, Robert Carson, Daniela Cassol, Stephen Chan, John Coleman, Kastan Day, Alejandro Duque, et al.

Novel Approaches Toward Scalable Composable Workflows in Hyper-Heterogeneous Computing Environments

Jonathan Bader
Berlin Institute of Technology
Germany

James Belak
Lawrence Livermore National Laboratory
USA

Matthew Bement
Oak Ridge National Laboratory
USA

Matthew Berry
University of Illinois
USA

Robert Carson
Lawrence Livermore National Laboratory
USA

Daniela Cassol
Lawrence Berkeley National Laboratory
USA

Stephen Chan
Lawrence Berkeley National Laboratory
USA

John Coleman
Oak Ridge National Laboratory
USA

Kastan Day
University of Illinois
USA

Alejandro Duque
Universidad San Francisco de Quito
Ecuador

Kjiersten Fagnan
Lawrence Berkeley National Laboratory
USA

Jeff Froula
Lawrence Berkeley National Laboratory
USA

Shantenu Jha
Rutgers University
USA

Daniel S. Katz
University of Illinois
USA

Piotr Kica
Sano Centre for Computational Medicine
Poland

Volodymyr V. Kindratenko
University of Illinois
USA

Edward Kirton
Lawrence Berkeley National Laboratory
USA

Ramani Kothadia
Lawrence Berkeley National Laboratory
USA

Daniel Laney
Lawrence Livermore National Laboratory
USA

Fabian Lehmann
Humboldt-Universität zu Berlin
Germany

Ulf Leser
Humboldt-Universität zu Berlin
Germany

Sabina Lichołai
Sano Centre for Computational Medicine
Poland

Maciej Malawski
Sano Centre for Computational Medicine
Poland

Mario Melara
Lawrence Berkeley National Laboratory
USA

Elais Player
Lawrence Berkeley National Laboratory
USA

Matthew Rolchigo
Oak Ridge National Laboratory
USA

Setareh Sarrafan
Lawrence Berkeley National Laboratory
USA

Seung-Jin Sul
Lawrence Berkeley National Laboratory
USA

Abdullah Syed
University of Missouri
USA

Lauritz Thamsen
University of Glasgow
UK

Mikhail Titov
Brookhaven National Laboratory
USA

Matteo Turilli
Rutgers University
USA

Silvina Caino-Lores
Inria
France

Anirban Mandal
University of North Carolina
USA

ABSTRACT

The annual *Workshop on Workflows in Support of Large-Scale Science (WORKS)* is a premier venue for the scientific workflow community to present the latest advances in research and development on the many facets of scientific workflows throughout their life-cycle. The *Lightning Talks* at WORKS focus on describing a novel tool, scientific workflow, or concept, which are work-in-progress and address emerging technologies and frameworks to foster discussion in the community. This paper summarizes the lightning talks at the 2023 edition of WORKS, covering five topics: leveraging large language models to build and execute workflows; developing a common workflow scheduler interface; scaling uncertainty workflow applications on exascale computing systems; evaluating a transcriptomics workflow for cloud vs. HPC systems; and best practices in migrating legacy workflows to workflow management systems.

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11.
https://doi.org/10.1145/3624062.3626283

CCS CONCEPTS

• **Computing methodologies → Distributed computing methodologies**;

KEYWORDS

Scientific workflows, large language models, workflow scheduling, high-performance computing, cloud computing, legacy workflows.

ACM Reference Format:
Jonathan Bader, James Belak, Matthew Bement, Matthew Berry, Robert Carson, Daniela Cassol, Stephen Chan, John Coleman, Kastan Day, Alejandro Duque, Kjiersten Fagnan, Jeff Froula, Shantenu Jha, Daniel S. Katz, Piotr Kica, Volodymyr V. Kindratenko, Edward Kirton, Ramani Kothadia, Daniel Laney, Fabian Lehmann, Ulf Leser, Sabina Lichołai, Maciej Malawski, Mario Melara, Elais Player, Matthew Rolchigo, Setareh Sarrafan, Seung-Jin Sul, Abdullah Syed, Lauritz Thamsen, Mikhail Titov, Matteo Turilli, Silvina Caino-Lores, and Anirban Mandal. 2023. Novel Approaches Toward Scalable Composable Workflows in Hyper-Heterogeneous Computing Environments. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3624062.3626283

1 INTRODUCTION

Scientific workflows have been almost universally used across scientific domains and have underpinned some of the most significant discoveries of the past several decades. As workflows have been adopted by a number of scientific communities, they are becoming more complex and require more sophisticated workflow composition and management capabilities across computing environments like edge, cloud and high-performance systems. The *Workshop on Workflows in Support of Large-Scale Science (WORKS)* has positioned itself as the primary venue for workflow researchers and developers to share and discuss innovative ideas to enhance the landscape of workflow research. Specifically, the lightning talks at WORKS provide a venue where the community can introduce works in progress, emerging technologies and frameworks, and workflow tools to lower the entry barrier and thus increase adoption of workflow-centric approaches. This paper provides a comprehensive overview of the five lightning talks from the 18th edition of the workshop (WORKS 2023), which cover the following topics:

Large Language Models (LLMs) (Section 2). The recent development of LLMs with multi-billion parameters, coupled with the creation of user-friendly application programming interfaces (APIs), has paved the way for automatically generating and executing code in response to straightforward human queries. This work explores how these emerging capabilities can be harnessed to compose complex scientific workflows, eliminating the need for traditional coding methods. This work presents initial findings from the integration of Phyloflow with OpenAI's function-calling API, and outlines a strategy for developing a comprehensive workflow management system based on these concepts.

Common Workflow Scheduler Interface (CWSI) (Section 3). Scientific workflow management systems (WMS) support the workflow execution and communicate with the infrastructures' resource managers. However, the communication between WMS and resource managers is complicated by inconsistent interfaces between WMS and resource managers and the lack of support for workflow dependencies and workflow-specific properties. To tackle these issues, previous work developed CWSI, a simple yet powerful interface to exchange workflow-related information between a WMS and a resource manager, making the resource manager workflow-aware. Prototype implementations show that the CWSI can reduce makespan up to 25% with simple workflow-aware strategies. This work shows how existing workflow resource management research can be integrated into the CWSI.

RADICAL Ensemble Toolkit (RADICAL-EnTK) (Section 4). When running at scale, modern scientific workflows require middleware to handle allocated resources, distribute computing payloads and guarantee a resilient execution. While individual steps might not require sophisticated control methods, bringing them together as a whole workflow requires advanced management mechanisms. In this work, the authors used RADICAL-EnTK—one of the SDK components of the ECP ExaWorks project—to implement and execute the novel Exascale Additive Manufacturing (ExaAM) workflows on up to 8000 compute nodes of the Frontier supercomputer at the Oak Ridge Leadership Computing Facility. EnTK allowed the authors to address challenges such as varying resource requirements (e.g., heterogeneity, size, and runtime), different execution environment per workflow, and fault tolerance. And a native portability feature of the developed EnTK applications allowed the authors to adjust these applications for Frontier runs promptly, while ensuring an expected level of resource utilization (up to 90%).

Transcriptomics Atlas pipeline (Section 5). Transcriptomics studies the RNA present in a specific cell or tissue at a given time or condition. This dependence on time makes the problem computationally challenging, as the data generated by transcriptomics experiments is larger than the genomics studies on DNA sequences. The goal of the Transcriptomics Atlas project is to create a database of analyzed RNA sequences corresponding to given tissue and organ types based on the data from public repositories and make it available for researchers. We describe our Transcriptomics Atlas pipeline as an example of a new data- and compute-intensive scientific workflow. After analysing the requirements of the tasks in the pipeline, we describe our proposed cloud architecture. We present the preliminary results of the experimental evaluation of the pipeline in the AWS cloud, and compare the performance results to the traditional execution on the HPC cluster.

JGI Analysis Workflow Service (JAWS) (Section 6). Legacy workflows often present numerous challenges, such as poor maintainability, reduced shareability, and non-portability. Centrally administered and supported WMS enable streamlined workflow design and execution within high-performance computing environments. Our group at the Department of Energy's Joint Genome Institute (JGI) has successfully migrated large, complex legacy workflows into a shared, modern WMS by leveraging the Workflow Description Language (WDL) to describe the workflow and containers to encapsulate the environment. The JGI has developed JAWS, a centralized workflow platform that integrates Cromwell and WDL with Globus file transport to run computational workflows across multiple HPC facilities, ensuring workflow portability, reusability, and shareability. This paper explores best practices, patterns, and anti-patterns in migrating manual or ad-hoc workflows to JAWS.

2 LEVERAGING LARGE LANGUAGE MODELS TO BUILD AND EXECUTE COMPUTATIONAL WORKFLOWS

By: *A. Duque, A. Syed, K. V. Day, M. J. Berry, D. S. Katz, and V. V. Kindratenko*

In this position paper, we argue that in the near future it will be possible to construct and utilize integrated scientific workflow description and execution systems using Natural Language within a chatbot-like environment, e.g., within the ChatGPT [35] framework. We observe that ChatGPT is capable of generating codes in various workflow description languages which then can be executed with the help of emerging plugins and new features, such as LangChain [5], function calling [37], Toolformer [40], code interpreter [36], etc. We believe that a next-generation workflow management system will simply provide a human-language-based interface to describe the work to be carried out, monitor its progression, and present the results, and will call underlying tools and methods to carry out a complex chain of computations in response. This will greatly simplify the process of applying complex computational pipelines by domain experts without any coding experience. To this end, we provide initial results with function calling API and propose a ChatGPT-based workflow management system.

2.1 Preliminary Results

As a demonstration, we used Phyloflow [16] as an existing workflow example and investigated how OpenAI's function calling API can be used to streamline the creation and execution of different tasks. The code for the work presented here is available in GitHub [28]. Phyloflow is a tool for performing phylogenetic tree computations that was initially developed in the Workflow Description Language (WDL) [13] and later ported to Python using the Parsl library [17]. The Parsl implementation consisted of a Parsl app for each task that corresponds to a data processing step within Phyloflow. A workflow is created by connecting Parsl apps through inter-application dependencies. It is important to note that for the first step of this Parsl workflow, we use physical files, and from there we work on data futures, which are promises of a file that will be generated by another running instance of a Parsl app (AppFuture).

Phyloflow employs WDL to perform calculations for phylogenetic analysis based on input data. This process is executed in multiple steps, beginning with the task 'vcf-transform.' VCF-transform receives a VCF (Variant Calling Format) file and extracts data from it, and transforms the data into the input format of 'pyclone-vi', which allows for mutation clustering calculations to be performed. Next, a TSV file containing the mutagenic data is created. The next task, 'pyclone-vi', processes the previously generated TSV file, performing the mutation clustering calculations. This yields clusters of mutations that share evolutionary relationships. The workflow subsequently reformats this data file for processing with SPRUCE (Somatic Phylogeny Reconstruction using Combinatorial Enumeration) in a separate workflow step. The final task, 'spruce-phylogeny', takes the SPRUCE-formatted TSV file as an input, and generates a JSON file that contains the computed information necessary to visualize the evolution of a tumor.

To enable the OpenAI function call API with Phyloflow, we created several functions that serve as adapters for Parsl apps. For each Parsl app, we created a *function_call_from_file*, which receives the paths to the physical files, and a *function_call_from_futures*, which receives the identifiers of the AppFutures on which the Parsl app depends. The difference between the two is that a *function_call_from_futures* first retrieves the AppFutures of the received IDs, and their DataFutures are extracted to be used as inputs. From there, the operation is identical: generate a new ID, generate a directory for outputs, run the ParslApp, index the AppFuture reference along with its ID in a global access dictionary and return the ID. This ID binding scheme with AppFutures was required to communicate with the OpenAI API.

Following the OpenAI specifications, we wrote function descriptions in JSON format for all of the *function_call_from_files* and *function_call_from_futures*. The communication scheme with the OpenAI API consists of sending this set of descriptions together with a natural language instruction prompted by the user. The job of the LLM is to determine which function needs to be executed to fulfill the statement, as well as the parameters to send to the function. By doing this, we were able to run individual Parsl applications within the workflow. However, what we really need is to chain the execution of several Parsl apps to generate complete workflow executions. This is where the notion of adding context and making successive API calls comes into play.

A predefined context is added, just like any other user message, that helps to better interpret any instruction. With this context and the user's message, the request to the API is made. The API responds with its choice of function to call. The function is executed, immediately returning the ID linked to the AppFuture. For the next API request, two new messages are added. The first message partially includes the previous response from the API, specifically the section of the message with the choice of the function to call is used. The second message is a new user message indicating the ID assigned to the newly executed Parsl app. By adding both messages, the AI understands which step it is in relative to the user's instructions and can also execute subsequent steps by having access to the scheduled AppFuture ID. This process is repeated until the stop flag is found in the API response.

Although the use of function calling has shown promise for executing workflows, our current implementation has at least two clear limitations. The first is that exceptions are not handled at the moment, which means that if the API executes a wrong function call, the program cannot recover from the failure. Optimally, the error should be forwarded to the API so that it can propose alternatives. The second limitation is that composing more complex workflows will eventually hit the token limit, for which there is no straightforward solution in the proposed scheme; we would need to invent a hierarchical schema for task decomposition.

2.2 Proposal for Next-Gen Workflow Engine

The system we discuss in §2.1 consists of two primary components: the execution of OpenAI API calls on OpenAI's servers and the processing stages of the Phyloflow application on our own servers. The current prototype sequentially executes these steps without much consideration for the results produced at each step. However,

a more advanced workflow engine should ensure two things: (1) The current step is executed as expected, free of errors or warnings, and produces the anticipated outcome; (2) The next step in the sequence can be executed given the outcome of the current step, the available computational resources, and other constraints.

We envision a workflow engine that accepts a high-level *description* of the work, provided in natural language. This description is then translated into multiple steps (a *plan*) based on available functional units such as executables or API calls. The engine then attempts to execute each step from the *plan*, taking into account hardware constraints such as the type of compute servers, available memory, storage size, while ensuring the task's completeness and correctness. If a task fails or the outcome is not as expected, the plan execution engine invokes a *debugger*. The *debugger's* role is to identify the issue so the task can be re-executed or the *plan* can be modified if necessary.

Figure 1 illustrates the overall structure of this approach. The *planner*, *executor*, and *debugger* are all AI agents that use LLM to process textual input, either to execute a task or to analyze and validate the execution results. A *human* operator may also be involved if the *debugger* cannot resolve the issue, or if there's a need to resolve ambiguities and make decisions.

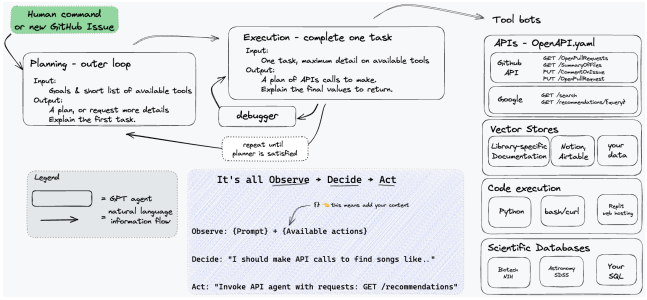

Figure 1: LLM agents collaborating to execute the workflow.

2.3 Conclusion

In §2.1, we successfully showcased a working prototype of a workflow engine that can execute a variety of tasks based on straightforward high-level user instructions. This accomplishment was made possible by leveraging OpenAI's function-calling API. This API infers tasks from the user's textual description and translates them into user-defined functions, which constitute the computational backbone of the actual scientific computing workflow. Furthermore, in §2.2, we outlined our ambitious vision for an advanced workflow execution engine. This engine is designed to manage complex, multi-stage workflows across an extensive computing infrastructure, all under the control of a natural human language interface.

3 THE COMMON WORKFLOW SCHEDULER INTERFACE: STATUS QUO AND FUTURE PLANS

By: *F. Lehmann, J. Bader, L. Thamsen, and U. Leser*

As workflows are becoming increasingly complex and datasets easily exceed hundreds of gigabytes or even terabytes [14, 34], scientists use scientific workflow management systems (WMS), such as Nextflow, Airflow, or Argo, and computer clusters. One essential feature of SMWSs is communication with a resource manager, such as SLURM, Kubernetes, or OpenPBS. Therefore, the WMS submits ready-to-run tasks to the resource manager, and the resource manager takes over the responsibility for assigning these tasks to a node that executes them. This simplifies the workflow execution on large-scale computing infrastructures and hides the complexity from the scientist. However, as with WMS, there is also a variety of resource managers available, and different clusters may use a different one. In a worst-case scenario, the WMS preferred by the scientist does not support the cluster's resource manager at all. Even if the WMS supports a given resource manager, features beyond submitting tasks and awaiting their completion are frequently not supported.

In this paper, we first give an overview of the Common Workflow Scheduler (CWS) and the Common Workflow Scheduler Interface (CWSI) which we both first presented in [31]. The CWSI is used to exchange workflow-related information between WMSs and resource managers. Second, we present prior results, showing promising outcomes when using the CWSI with workflow-aware resource management methods. Moving on, we outline WMS, where we started to implement CWSI support and demonstrate how the CWS can serve as a central point for provenance. Last, we illustrate how the CWS can be extended with new scheduling, resource allocation, and runtime prediction methods.

3.1 Common Workflow Scheduler

To address the challenge that resource managers schedule workflow tasks without workflow awareness, we developed the Common Workflow Scheduler (CWS) [31]. The CWS allows for the transfer of essential information, such as input files, CPU, and memory requests, along with task-specific parameters using the Common Workflow Scheduler Interface (CWSI). Task-specific parameters vary for each task invocation and are passed on to the utilized tools.

In Figure 2, we provide an architectural overview for a single resource manager, in this case, for Kubernetes. The CWS runs as a component in the resource manager and exposes the CWSI. A resource manager has to implement the CWS with its interface once. Conversely, a workflow engine needs to implement support for CWSI to work with all resource managers already offering CWSI. WMSs such as Airflow, Nextflow, or Argo send their requests, which are then kept in memory of CWS. From this storage, the CWS can fetch the workflow graph and task dependencies and use this information for scheduling. This storage can further be used for provenance to trace the workflow execution; we elaborate on this in Section 3.3. The CWS can be extended with task runtime and resource predictors that read task information from the storage and learn characteristics. Such learned characteristics can then be used to predict the demands for upcoming tasks, which is helpful for better scheduling. We provide examples for such prediction strategies in Section 3.4. Notably, workflow engines with CWSI support do not need their own scheduler component. Instead, all ready-to-run tasks are submitted to the resource manager and the

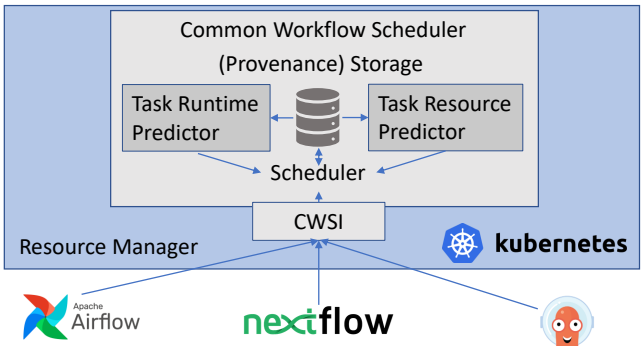

Figure 2: Architecture overview: The Common Workflow Scheduler with its interface and task runtime and task resource predictor component for Kubernetes as an exemplary resource manager.

scheduling happens there. We have implemented a plugin[1] for the WMS Nextflow to communicate with the CWSI and the CWS for the resource manager Kubernetes[2].

3.2 WMS support

We started by implementing the CWSI for a resource manager, Kubernetes, and a WMS, Nextflow. We are now actively working to extend our project to support other popular WMSs, namely Airflow and Argo, to further explore and demonstrate the benefits of the CWSI. Below, we describe these three WMSs and discuss the integration of the CWSI.

Nextflow is a workflow engine initially designed for bioinformatics but getting uptake also in different domains and is used by more than 1,000 organizations [26, 32, 44]. One of the main advantages of Nextflow is its support for, at the time of writing, 20 different resource managers. The large support makes it easy to port Nextflow workflows between environments. The support is achieved by abstracting the resource manager from the scientist but also from the internal Nextflow logic. Accordingly, Nextflow only supports the basic features of resource managers. For example, on SLURM, the task dependency feature is not used. Thus, Nextflow can profit from providing additional workflow context to the resource manager.

Airflow is an Apache Incubator project designed for workflow management. Similar to Nextflow, Airflow supports Kubernetes as a resource manager and is also not exclusively tied to it. Airflow supports workflow-aware scheduling for Kubernetes through a tailor-made strategy exclusively implemented for the Airflow-Kubernetes interplay. Therefore, Airflow starts a big worker on every node for the whole workflow execution and assigns tasks into these worker pods bypassing Kubernetes' task assignment logic. However, this strategy has a significant drawback: the big containers will request resources for the entire workflow execution time regardless of the actual load. As many workflows have a merge point somewhere, where the entire execution is waiting for one particular task, this strategy leads to substantial resource wastage. By integrating the CWSI into Airflow, we aim to retain its workflow-aware scheduling capabilities while preventing unnecessary resource requests throughout the runtime. This optimization ensures more efficient utilization of resources and minimizes wastage on a large scale. One big difference to our already existing Nextflow interaction is the knowledge of the physical DAG in Airflow. While this was foreseen in the development of the CWSI, we have to make use of it in our CWS implementation.

Argo is a WMS designed exclusively for Kubernetes. However, since Kubernetes lacks support for task dependencies, Argo also submits each task individually, and Kubernetes then schedules them in a FIFO manner. This is comparable to the strategy of Nextflow and, thus, makes Argo an ideal candidate to support our CWSI. Just like Nextflow, Argo is expected to benefit in a similar way. We are currently working on developing an Argo extension to achieve this.

3.3 Provenance with the CWSI

Workflow provenance is one aspect that needs to be addressed in all WMS [14, 20, 25]. Since the CWSI takes a central role in workflow execution, possessing comprehensive knowledge of the resource manager and the WMS, it emerges as the most suitable entity for the management of provenance data.

All WMS represent provenance differently, so it is very heterogeneous [24]. Further, resource managers and WMSs are only designed to gather a portion of the available data, each focusing on collecting data in its own scope [20]. Accordingly, the resource manager traces the node states while the WMS collects task-related metrics. The CWSI is particularly implemented for each resource manager and can support a resource manager's specific APIs to collect traces while it has knowledge about the workflow. By gathering and storing all metrics and task dependencies in a centralized manner, provenance becomes more streamlined and manageable.

Another significant advantage of using the CWSI for provenance is that the data will be available across different WMS, even if a particular WMS does not yet provide built-in provenance data. This interoperability ensures that provenance information can be maintained consistently and comprehensively, enhancing workflow traceability and reproducibility. In turn, researchers and scientists can have greater confidence in the reliability and trustworthiness of their results.

3.4 Advanced Resource Management with the CWSI

As we saw in the previous section, the CWS provides information about task executions and performance metrics. Using this information allows possible interface extensions to derive task characteristics from it. Task characteristics can be predicted runtime, CPU or memory usage, which can be used for scheduling and fed back to the WMS. Many scheduling strategies, such as HEFT [45], require knowledge of this.

The CWSI provides information to train task resource prediction models, e.g., the number of file inputs, input sizes, or peak memory, which are retrieved and stored from monitoring. As these metrics are constantly gathered and updated, also online learning approaches are applicable. Therefore, we plan to integrate existing task resource prediction methods in our CWSI prototype to a)

[1] https://github.com/CommonWorkflowScheduler/nf-cws
[2] https://github.com/CommonWorkflowScheduler/KubernetesScheduler

increase workflow performance and b) evaluate them under real-world conditions.

Since Lotaru [18] and other research approaches that support heterogeneous infrastructures to predict task runtimes require machine characteristics, we are extending our CWSI to store such information and extend the prototype to gather these metrics with Kubestone[3]. We are currently incorporating Lotaru into the CWSI prototype to handle unknown workflows or workflows with a lack of historical data. Further, we plan to implement other research methods that perform better with more training data provided by the provenance store of CWSI.

The CWSI, together with task runtime and resource prediction, provides additional information to apply more sophisticated workflow scheduling techniques. We are currently implementing the Tarema [19] strategy into our CWSI prototype and plan other more sophisticated approaches enabled through the additional data provided by the CWSI and their plugins.

3.5 Conclusion

In this paper, we presented the status quo of the Common Workflow Scheduler Interface and described the available plugin for Nextflow and the integration into Kubernetes. Further, we have demonstrated that by implementing the CWSI alongside basic scheduling approaches like rank and file size, we achieve an average runtime reduction of 10.8%. Next, we outlined upcoming support for the workflow engines Airflow and Argo and how to extend the storage to become the central place for workflow provenance. Additionally, we presented our next steps to implement resource allocation, runtime prediction, and new scheduling methods. We assume that the planned workflow algorithms that consider cluster heterogeneity and task runtime, as we outlined in this paper, will further improve resource efficiency.

4 SCALING ON FRONTIER: UNCERTAINTY QUANTIFICATION WORKFLOW APPLICATIONS USING EXAWORKS TO ENABLE FULL SYSTEM UTILIZATION

By: *M. Titov, R. Carson, M. Rolchigo, J. Coleman, J. Belak, M. Bement, D. Laney, M. Turilli, and S. Jha*

The Exascale Additive Manufacturing project (ExaAM) [47] has developed a suite of exascale-ready computational tools to model the process-to-structure-to-properties (PSP) relationship for additively manufactured metal components. ExaAM built an uncertainty quantification (UQ) pipeline to quantify the effect that uncertainty has on local mechanical responses in processing conditions. The UQ pipeline consists of 3 main stages, and each stage is represented with a corresponding workflow.

The UQ pipeline imposes certain requirements on the ensemble management tools: (i) ability to centralize a streamline of the whole pipeline; (ii) control different resource requirements (e.g., either having one large batch job for all workflows or setting a workflow per batch job with the different numbers of acquired compute nodes and runtime); (iii) support different heterogeneous high performance computing (HPC) platforms (including different system architectures); and (iv) fault-tolerance of the tools and executing processes (i.e., computing tasks that represent batch job steps).

In response to the stated requirements, a corresponding workflow management toolkit RADICAL-EnTK (Ensemble Toolkit) [21]—part of the ExaWorks Software Development Kit (SDK) [43]—was evaluated and chosen. EnTK provides the possibility to: (i) automate runs of different workflows together, while providing an isolated execution environment per each workflow; (ii) control the execution state of a workflow and its every task individually; and (iii) handle the size of a workflow dynamically, e.g., create a new workflow stages based on the status of previously executed stages.

4.1 RADICAL Building Blocks

RADICAL-Cybertools (RCT) [46] are software building blocks designed to develop efficient and effective tools for scientific computing. Specifically, RCT allow one to develop scientific applications with up to 100,000 heterogeneous computing tasks (i.e., a self-contained process, Python function, or executable, and can be independently executed from other "tasks") and executing them on the largest HPC platforms in the world at unprecedented scale.

RCT enable writing workflow applications with *task-*, *resource-* and *platform*-level heterogeneity. Each building block is designed to work as a standalone system or integrated with other tools from RCT, or third-party software tools. Currently, RCT integrate with other ExaWorks [15] SDK components.

Two of the most commonly used building blocks are RADICAL-Pilot (RP) [33] and EnTK. RP is a pilot-enabled runtime system that allows users to concurrently execute up to 10^4 heterogeneous computing tasks on up to 10^5 heterogeneous resources. RP manages concurrent and sequential execution of single/multi core/GPU/node non/MPI computing tasks on HPC platforms. EnTK is a workflow engine designed to support the coding and execution of scientific workflows represented using the PST model. EnTK PST stands for Pipeline-Stage-Task, where *Pipeline* is a sequence of *Stages*, and each *Stage* is a set of independent computing *Tasks*. Multiple pipelines can be executed concurrently, while stages, within each pipeline, are executed sequentially. Grouping tasks into stages represents dependencies among tasks and enables the concurrent execution of tasks from the same stage. EnTK utilizes RP to acquire and manage HPC resources, and place and launch tasks on those resources.

4.2 ExaAM Workflows

While Fig. 3 shows the full ExaAM UQ pipeline, we will focus on its three main stages.

Stage 0 generates the UQ grid using TASMANIAN [41] and then all the necessary directories and their input files for *Stage 1*.

Stage 1 can be represented as two sub-stages, where thermal melt pool simulations are run in one sub-stage (AdditiveFOAM [23, 30], an extension of OpenFOAM for AM processes) and the microstructure generation is run in a subsequent sub-stage (ExaCA [38]). *Note* that AdditiveFOAM includes CPU-only computing tasks and requires even and odd runs to generate all melt pool thermal histories. These runs have an associated post-processing step to gather all the necessary output files into a single file for the following

[3] kubestone.io

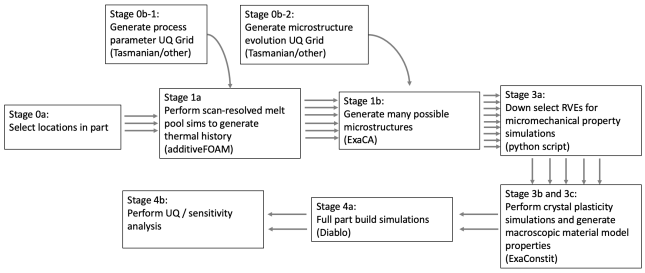

Figure 3: Extended schematic representation of the ExaAM UQ pipeline.

sub-stage ExaCA. ExaCA includes CPU-GPU computing tasks. The microstructure is generated for the cartesian product between all the thermal pool cases and different microstructure UQ parameters. Once those runs have completed, a post-processing step occurs to prepare data for the local property calculations.

Stage 3 (local property calculations) runs all the ExaConstit [22] simulations. It is driven by one Python script, which reads in all the generated microstructures; coarsens the microstructures; and generates all the simulation option files and directories associated with all the different loading directions, temperature cases, and representative volume elements (RVEs) from ExaCA microstructures. The script then has a number of built-in job scheduler backends (e.g., Flux, LSF) available and can run/submit all the jobs in one go. Once all the simulations are complete, an optimization script then calculates the necessary macroscopic material model parameters to be used in full part-builds.

We implemented the UQ pipeline as a set of EnTK workflow applications, where each UQ stage corresponds to the EnTK application (with a single EnTK pipeline) and consists of one or more EnTK stages. We implemented the pre-/post-processing operations into dedicated EnTK tasks and grouped them into EnTK stages. Having a dedicated application per UQ stage allows us to execute the stages individually or as part of the whole UQ pipeline. The developed code is located in the project's GitHub repository [42].

UQ Stage 1 is transformed into the EnTK application with the following EnTK stages: AdditiveFOAM's pre-processing and AdditiveFOAM, ExaCA and ExaCA-Analysis. The various melt pool cases (AdditiveFOAM) and microstructure generation cases (ExaCA) are represented as single tasks within each of their corresponding EnTK stages. All of the logic needed to drive Stage 1 is implemented in one script, which acts as a standalone EnTK application. This application handles failed tasks by re-submitting them as part of the consecutive batch job (i.e., the next EnTK run). This automated process helps to deal with hardware failures to run collected failed tasks using a new job allocation. During re-submission of failed tasks, the execution order is preserved according to the order of the original EnTK stages.

UQ Stage 3 integrates a corresponding EnTK application to leverage all the job ensembles (i.e., set of simulations per batch job).Additional logic has been added so that each ensemble respects Frontier's job scheduling policy in terms of walltime limits per amount of requested compute nodes. Each simulation is represented as a single EnTK task. Task failures are handled following the same approach as for UQ Stage 1 (re-submitted job size is smaller and correlates to the number of failed tasks).

Using EnTK allowed us to abandon the manual creation and management of batch scripts in favor of having a single ensemble manager EnTK to handle everything in one large job or subsequent smaller jobs submissions. We also introduced fault-tolerance for task execution level, which improved the efficiency of each EnTK application and UQ pipeline as a whole.

4.3 Frontier run

We used Crusher (an early-access testbed platform for the Frontier system) to evaluate the upcoming exascale system's architecture, and to pre-configure and adjust RP components. Developed EnTK applications are easily reconfigured for each platform via its resource configuration and corresponding execution environment setup for every task type. These applications have been tested on multiple platforms at OLCF with different job schedulers.

Early runs on Summit and Crusher utilized up to 10 compute nodes for several hours, and were used to verify the correctness and stability of the execution process before targeting Frontier. With the scale-up on Frontier, resource utilization is the following:

- AdditiveFOAM workflow utilized 40 compute nodes for 2 hours (every task requires 4 nodes with 56 cores per node);
- ExaCA workflow utilized 125 compute nodes for 4 hours (every task requires 1 node and utilizes 8 MPI ranks with `7CPUs-1GPU` decomposition);
- ExaConstit workflow utilized 8000 compute nodes (85% of Frontier's nodes) for up to 3.3 hours to run and orchestrate 7875 tasks (every task requires 8 nodes with 8 MPI ranks per node with the typical `7CPUs-1GPU` decomposition, and runtime ~10-25 min).

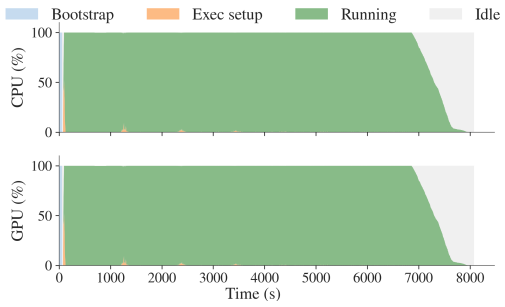

Figure 4: Resource utilization by the EnTK application (UQ Stage 3): 100% corresponds to 448,000 CPU cores (not considering 8 cores per node reserved for system processes) and 64,000 GPUs.

"Full" scale run for UQ Stage 3 constantly utilized most of the available resources (total resource utilization is 90%) with a minimal overhead (OVH) of the EnTK application (i.e., bootstrapping EnTK components). Fig. 4 shows that OVH (light blue color) is just 85s, while the total execution time of all simulations (TTX) is 7989s (the job runtime is 8074s). ExaAM workflows implemented with EnTK reached a scheduling throughput of 269 tasks/s, launching

51 tasks/s. Those rates are part of Fig. 5 (initial slopes of blue and orange lines), which also shows the number of tasks executing concurrently (orange color) as well as the number of tasks pending to be launched (blue color).

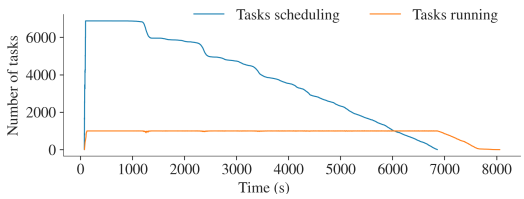

Figure 5: Concurrency of 7875 EnTK tasks (UQ Stage 3) in scheduling and running (execution) states.

We registered only 10 task failures across the UQ Stage 3 run. Two tasks failed on the very last simulation step due to too large of a time step for the specific loading condition and RVE, but they were still far enough out for the purpose of constructing the macroscopic material model parameters. The other eight tasks failed due to a single node failure and ran successfully once automatically resubmitted on Frontier by EnTK.

4.4 Conclusion

In this paper, we have presented the novel ExaAM UQ pipeline and its implementation with RADICAL-EnTK. Running simulations for the additive manufacturing process requires an exascale heterogeneous machine (for CPU-GPU intensive computations), as well as an efficient ensemble manager to automate the execution of the whole UQ pipeline and manage varying resource requirements, different execution environments while offering fault-tolerance. Selecting EnTK as a workflow engine ensures running the UQ pipeline efficiently and effectively. Once implemented in EnTK, these applications are portable across multiple LCF platforms. We tested them on OLCF Summit, Crusher and Frontier. For runs on Frontier, we progressively increased scale until executing the ExaConstit workflow with 7875 tasks on 8000 compute nodes (85% of Frontier's nodes). We achieved a total resource utilization of 90%.

5 TRANSCRIPTOMICS ATLAS PIPELINE: CLOUD VS HPC

By: *P. Kica, S. Licholai, and M. Malawski*

Transcriptomics is an interdisciplinary scientific field that focuses on studying the complete set of all RNA molecules present in a specific cell or tissue at a given time, which is referred to as the transcriptome. Transcriptomics methods, such as RNA-seq, enable the identification, quantitative analysis, and characterization of various types of RNA, including mRNA, rRNA, tRNA, and non-coding RNAs [29]. The aim of this work is to provide a Transcriptomics Atlas that will allow more precise experimental design even before the laboratory testing stage.

Given the wide range of potential uses for the RNA-seq data in this project, we have chosen to implement two different pathways: a faster one: *Salmon Pipeline*, and a more accurate one: *STAR Pipeline*. The first one is based on pseudo-alignment and the Salmon tool [8], and we anticipate that it will provide a suitable output for experiments involving quantitative expression analysis of individual genes. The other is based on alignment and the STAR tool [10, 27] and allows, in addition to quantification, the analysis of splicing variants or non-canonical forms of RNA.

The goal of this paper is to present the technical details of our Transcriptomics Atlas pipeline. We describe the cloud architecture design for execution of the pipeline on AWS cloud. We present the results of our experimental evaluation of the pipeline in the AWS cloud using a sample data set, and compare the performance results to the traditional execution on the HPC cluster.

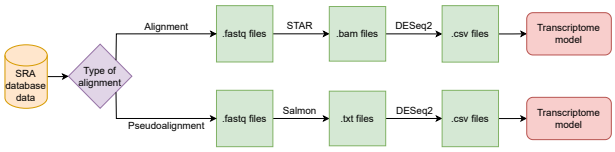

Figure 6: Transcriptomics Atlas pipeline.

5.1 Transcriptomics Atlas Pipeline

To build the Transcriptomics Atlas for a given cell or tissue type, the prerequisite is to obtain a set of identifiers of the input files (usually 100s of 1000s of files are needed per cell/tissue type) from the the NCBI database [7]. Then, for each input id, the full pipeline needs to be executed, each pipeline consisting of the steps described below. Therefore, the high-level workflow consists of multiple independent pipelines processed in parallel. The current implementation is available in Sano's Github repository[12].

Pipeline description Original pipeline was created for HPC execution by a domain specialist. It consists of four steps: (1) downloading .sra file using *prefetch* tool; (2) converting into .fastq files using *fasterq-dump* tool; (3) alignement and quantification of reads using *Salmon* [8] or *STAR* [10] [27]; and (4) count normalization using *DESeq2* [3]. Both prefetch and fasterq-dump are provided by *SRA-Toolkit* [9]. The general pipeline is presented in Fig. 6 with alignment (STAR) and pseudoalignement (Salmon) paths respectively. In this work we focus on Salmon path, as the STAR path is part of future work.

Pipeline steps resource requirements The main requirements are given by the STAR and Salmon software. Both require an index which will be used in alignment step. In the case of Salmon the generated index on human transcriptome is about 1GB in size, but in case of STAR the index is generated on human whole genome and is much bigger - 90GB. The index generated is affected mostly by the size of the genome/transcriptome and resulting files are reused among the future pipeline executions. STAR also requires significant amount (in this case over 250GB) of RAM available - most of all to load the given index. Salmon, on the other hand is much less demanding as it is possible to process most of the SRA files using only 2 cores and 8GB of RAM. All other pipeline steps do not exceed the given resource values.

Input data The input data for this pipeline are *.sra* files which contain RNA sequencing data. They are hosted on NCBI [7] and

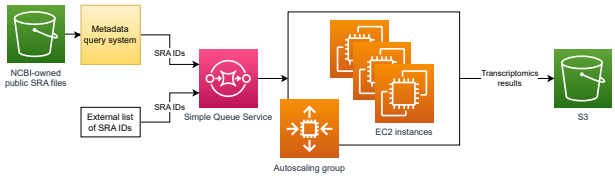

Figure 7: Transcriptomics Atlas cloud architecture.

publicly available in multiple ways such as download using SRA-Toolkit, direct access to buckets on either AWS S3 or Google Cloud Storage. The specific SRA IDs which have to be processed are identified by properties such as tissue name and origin, RNA sequence, and sequencer type. The total size of the SRA files that need to be processed for the atlas consisting of 20 human tissues is 8.6TB.

Connecting the steps and EC2 deployment In order to deploy the pipeline to the EC2 instance, it was necessary to install and configure all the software and tools used in the pipeline, followed by the creation of Amazon Machine Image (AMI). Then each step of the pipeline was connected with a Python script that handles pipeline execution order (using *subprocess.run()* method). This approach was extended further to allow sending execution logs to CloudWatch, listen for messages on SQS queue, and send results and execution metadata to S3 buckets (using *boto3* library). All deployment was automated using Terraform [11] to allow easy infrastructure management and reproducibility. To monitor the execution of each pipeline step on EC2 instances we used CloudWatch Agent [2] software with *procstat* plugin which allows to monitor per-process metrics. It allows to gather metrics such as usage of CPU, memory, disk and network.

Cloud Architecture An important step before designing the cloud architecture is to gather resource requirements for each step of the pipeline. Based on the tools' documentations, cloud services limits, experimental pipeline execution resource consumption and the data provided by the genomics specialist, the suitable approach seemed to use EC2 instances as the main compute service. The full cloud architecture for the Transcriptomics Atlas pipeline is presented in Fig. 7. All the generated intermediate files (e.g. .fastq) are not needed after the pipeline has finished. Also alternative approaches that would require splitting the pipeline steps among services incur heavy transfer and communications costs. Therefore each SRR file is processed on a single EC2 instance from start to finish of the pipeline. We use Auto-Scaling Group in order to automatically scale the number of instances. The final results are uploaded to an S3 bucket. This is a cloud-native approach and easy-to-scale architecture.

Pipeline Containerization for HPC In order to execute several instances of *Salmon Pipeline* on HPC the best approach is to containerize the pipeline and start multiple jobs with the container. The Docker [1] image created for this purpose consists of i.a. Salmon, preconfigured SRA-Toolkit, Python and R with dependencies (e.g. DESeq2), and Salmon index (1GB in size). Such an image can be deployed for serverless computing services due to sufficient resource requirements in contrary to the *STAR Pipeline*. After creating the Docker image we can upload it to DockerHub [4]. The containerization software used on the Ares cluster is Apptainer which can pull Docker images from remote repositories and translate them into compatible (*.sif*) images. One of the limits of the Apptainer software is that the container cannot be modified so in order to create or modify files we need to mount external writable directories (e.g. *SCRATCH*) for intermediate pipeline files or use the Persistent Overlays feature. In the future case of *STAR Pipeline* the approach will need a modification as the index required by STAR tool is 90GB in size thus it seems better to e.g. make the index available on *SCRATCH* partition and mount it to each container.

5.2 Experiments

The goal of the experiments was to test the correctness of the pipeline implementation, to validate our cloud deployment, and to compare the performance of the cloud version to the HPC version.

5.2.1 Salmon Pipeline Results. The experiment with EC2 instances delivered insightful metric results which are presented in summary in Table 1. All 99 files were processed in about 2.7h and not a single step failed during the execution. We can confirm that Salmon is indeed the most resource-consuming step. The high mean of CPU iowait in the fasterq-dump step suggests increasing the performance of the attached EBS. As we can see none of the steps exceed 4GB in RAM usage which may be a reason to use compute-optimized instance (e.g. c6a.large type which has 2vCPU and 4GiB RAM) for slightly more cost-efficient computation unless some bigger .sra files will require more than 4GiB of memory.

The HPC experiment took 2.5h to complete and no failure occurred on any step. Per-file comparison in execution times has been carried on between Cloud and HPC and the results are presented in Table. 2. The Salmon and fasterq-dump steps present relatively better performance on HPC. However we can clearly identify that prefetch is faster on AWS - this can be the case because we use the "report-cloud-instance-identity" in SRA-Toolkit config which means that we most likely download directly from S3 buckets and through the AWS backbone network instead of going through the Internet. The reported job efficiency for the experiment was about 72%.

5.3 Conclusion

Based on the results we are able to compare execution times between Cloud and HPC, identify strengths of each environment and better understand the pipeline. We have created a cloud architecture that fits requirements. On top of that we can monitor the EC2 instances resource usage in each step. The Salmon Pipeline was deployed to the HPC using Apptainer containers. We have identified places for further improvements and fine-tuning of the pipeline. Both environments have their advantages for the creation of the Transcriptomics Atlas.

The next step in this research is to create the more CPU- and memory-intensive STAR Pipeline and perform similar or larger experiments on HPC and Cloud. Also it can be beneficial to test the pipelines on other supercomputers as well as to deploy Salmon Pipeline to serverless computing services (e.g. AWS Elastic Container Service with Fargate launch type). Integrating Lithops [39][6] for workflow management can make the solution more portable and robust. Interesting architecture may be obtained with hybrid approach where we split the workload among HPC and Cloud.

Table 1: Aggregated "instance-wide" metrics during execution of each pipeline step. Baseline memory is approx. 300MB.

Pipeline step	Prefetch		Fasterq-dump		Salmon		DESeq2	
	mean	max	mean	max	mean	max	mean	max
CPU usage	21%	70%	56%	94%	94%	100%	39%	59%
CPU iowait	3.7%	47%	26%	91%	1.5%	90%	3.4%	47%
MEM usage	323MB	410MB	394MB	760MB	840MB	2.8GB	532MB	1GB

Table 2: Performance comparison between Cloud and HPC. Calculated as an average of relative difference in execution time.

Pipeline step	Prefetch		Fasterq-dump		Salmon		DESeq2	
	mean	max	mean	max	mean	max	mean	max
Cloud execution times	0.6min	3.9min	1.4min	5.7min	9.6min	43min	11s	36s
HPC execution times	2.1min	19.6min	0.8min	3.5min	8min	34.1min	10s	12s
HPC performance	87% slower		30% faster		19% faster		No difference	

6 PATTERNS AND ANTI-PATTERNS IN MIGRATING FROM LEGACY WORKFLOWS TO WORKFLOW MANAGEMENT SYSTEMS

By: *D. Cassol, J. Froula, E. Kirton, S. Sul, M. Melara, R. Kothadia, E. Player, S. Sarrafan, S. Chan, K. Fagnan*

A significant challenge with legacy workflows lies in their lack of maintainability, shareability, and portability. Custom workflows require a significant amount of code to orchestrate task execution (e.g., "pipeline glue code"), and they often operate within a unique user environment, making it difficult to ensure consistency and reproduce results when shared among different users. To effectively navigate this landscape, the Joint Genome Institute (JGI) adopted a Scientific Workflow Management System (WMS) that depends on a standardized Domain Specific Language (DSL) for workflows. The workflow DSL is designed to easily express the actions and relationships specific to running workflows, making it more suitable for describing workflows than general-purpose languages such as Perl, Python, or Java.

This paper explores the best practices, patterns, and anti-patterns associated with porting manual or ad-hoc workflows to automated WMSs, focusing on popular WMS such as Cromwell [48] and WDL. By following these guidelines, users can streamline their processes, enhance productivity, and realize the full potential of automated workflow management systems. Additionally, we highlight the creation of the JGI Analysis Workflow Service (JAWS), a centralized workflow platform developed by JGI.

6.1 Common Strategies for Migration to Workflow Management Systems

Modularization Modularization involves the decomposition of complex workflows into smaller, more manageable tasks, making migration and testing more straightforward. This strategy promotes the creation of workflows that are easy to maintain and expandable. Additionally, a modular framework assists in pinpointing bottlenecks and potential areas for refinement. For instance, separating tasks based on distinct resource requirements is crucial to ensure efficiency, resulting in both time and cost savings. Most workflow managers can efficiently handle fault-tolerance, task interruptions, workflow recovery, and detect when an identical task has been run in the past and avoid re-computing the results. By modularizing workflows, we can leverage these features. On the other hand, how individual commands are organized into tasks has a potentially huge impact on efficiency, reducing the strain on the filesystem and minimizing overhead. For instance, in one of JGI's workflows, by integrating four separate tasks into a single task, we cut the execution time by 70% and decreased the number of shards by 71%.

Containerization. Containerization is a common practice for DevOps and cloud-based architectures, and therefore, adopting it for scientific workflows is an obvious benefit. This reduces the need for manual processes, including source code compilation, manual version tracking, and writing and maintaining custom code. As an example, JGI workflows can use the same container image across the Perlmutter, Tahoma, Dori, and Lawrencium clusters, as well as in Amazon Web Services (AWS).

Workflow Provenance. Tracking of changes in software versions, parameters, etc. is key to enabling reproducibility. Workflow managers automate this process, maintaining uniformity in data formats, naming conventions, and directory structures. This approach fosters better maintainability for the author and a smoother transfer of the code to others. By aligning with established standards, researchers can mitigate discrepancies, minimize errors, and confidently reproduce outcomes, thereby enhancing the reproducibility of their data analysis.

Comprehensive Documentation and Community Development. When porting from a legacy workflow, it is crucial to create detailed documentation of the original manual workflow and

provide guidelines on how to use the new version. This documentation not only acts as a reference but also makes it easier for the community to repurpose the workflow.

Performance Metrics Collection. A centralized workflow service presents an opportunity to collect performance metrics for all workflows executed across the organization. By using tools such as psutil and Elastic Stack, these performance metrics can give valuable feedback on resource usage, pointing to problematic areas in the workflows themselves or bottlenecks in the workflow management system. This approach enables accurate selection of the amount of memory and number of CPUs, and also provides insight into I/O patterns to be taken into account when scheduling workflows for execution.

6.2 Anti-patterns to Avoid

Migrating Complex Workflows. Trying to port an entire legacy workflow in a single attempt can be challenging and may not always be the most efficient approach. Such endeavors frequently lead to overlooked or unidentified errors, making the debugging process significantly longer and more convoluted. Instead of a blanket migration, a phased and systematic approach might be more conducive, ensuring that each workflow segment is seamlessly integrated into the new system, thereby minimizing potential hitches and enhancing the transition's overall efficiency. This highlights the desire to modularize workflows, enabling individual segments to transition into WDL while maintaining productivity.

Neglecting Version Control. Overlooking the use of version control systems, such as Git, can present significant challenges when developing or porting a legacy workflow. Without a robust system to monitor modifications and trace alterations, being able to pinpoint when and where they were made becomes increasingly difficult. Additionally, the absence of repository management further exacerbates these challenges. Proper repository management ensures organized storage, easier collaboration, and a streamlined method to revert or retrieve specific versions, enhancing overall efficiency and reproducibility. In addition, by using version sha256 on container images for running workflows, it is possible to be very precise about the software's version used to execute workflows.

Inadequate Testing and Validation. Neglecting adequate testing and validation during the transition from legacy pipelines introduces considerable potential risks. Existing workflows should feature a small demo dataset specifically designed for integration tests. By adopting a test-driven development, one ensures that the new workflow's migration and development are reproducible, simplifying the process of identifying and rectifying any issues during both the developmental and testing phases. Testing new changes to the pipeline can even be automated via Git's CI/CD pipelines.

Inappropriate Parallelism. Task parallelism involves distributing tasks across independent compute nodes, primarily when no data dependencies exist between tasks. For example, Cromwell uses the scatter function from the WDL standard library, which will produce parallel jobs that run a series of identical tasks on different inputs. However, inappropriately parallelizing tasks can lead to a series of issues. This becomes particularly problematic if the computation task requires significant I/O filesystem operations, resulting in unwanted overhead. Achieving optimal performance in workflow management requires a careful balance. On one hand, there's the time and resources dedicated to each parallel task's execution, and on the other, there's the overhead in the filesystem for managing these tasks. It is advisable that each parallel job should have a minimum runtime of 30 minutes.

Unconstrained Task Parallelism for Shared Cluster Resources. Traditional batch scheduling systems usually implement fair share policies to prevent users from monopolizing a shared computing resource. Legacy workflows often submit directly to a batch scheduler under their own account, in which case fair share is handled by the scheduler. Tools such as Cromwell have their own interfaces into execution engines and may use a shared account for all users who run their workflows. Cromwell does not implement fair share policies, as a result, users sharing a single Cromwell service may find that a single user has been able to monopolize all resources allocated to Cromwell, for example, with a highly parallel task. When working with WMS, it is important to explicitly review the parallelism constraints that are configured within the WMS and configure them to support fair share within the WMS.

6.3 Centralized Workflow Management

With access to several DOE computing resources as well as Amazon Web Services, we need to guarantee effortless portability of workflow analysis across a range of resources. To address this, we have developed JGI Analysis Workflow Service (JAWS) as a centralized framework to run computational workflows across multiple DOE resources. JAWS uses Globus and AWS S3 protocol to transfer data and code to user-specified compute resources, subsequently executing the computation by leveraging the Cromwell engine for execution of WDLs and returning the results. Additionally, JAWS uses containers (compatible with Docker, Shifter, and Singularity) to ensure portability across computing environments and workflow task codebases. To run the compute tasks, JAWS uses HTCondor as the backend to Cromwell, which can manage workers on an HPC cluster. Adopting workflow managers to route jobs and data across multiple sites seamlessly facilitates the portability, reusability, and shareability of the analysis workflows.

6.4 Conclusion

The journey from manual workflows to centralized, automated workflow systems, while promising, can be strewn with challenges. Transitioning to WMSs offers a strategic approach to optimize computational workflows by ensuring efficient resource allocation, improved reproducibility, and enhanced scalability. Using a standardized workflow language and fostering workflow sharing can significantly diminish inefficiencies in analysis, in both computational and human resources. It's essential to identify successful patterns and avoid common pitfalls for a smooth transition from isolated operations to collaborative and FAIR workflows. Such a shift not only enhances performance and supports FAIR principles but also fosters a collaborative culture using a unified workflow platform.

ACKNOWLEDGMENTS

The materials this work is based upon are supported by NSF award 2050195; the German Research Foundation (DFG) CRC 1404: "FONDA:

Foundations of Workflows for Large-Scale Scientific Data Analysis;" the Office of Science of the U.S. Department of Energy (DOE) under Contract No. DE-AC05-00OR22725; the Exascale Computing Project (ECP, 17-SC-20-SC) ExaWorks project under DOE Contract No. DE-SC0012704; Lawrence Livermore National Laboratory under Contract No. DE-AC52-07NA27344; DOE INCITE awards for allocations on Summit, Crusher, and Frontier; the European Union (EU) Horizon 2020 research and innovation programme under grant agreement Sano No. 857533 within the International Research Agendas programme of the Foundation for Polish Science, co-financed by the EU under the European Regional Development Fund; the EU Horizon 2020 research and innovation programme under grant agreement NEARDATA No. 101092644; the PLGrid Infrastructure and the Ares supercomputer at ACC Cyfronet AGH; and DOE Joint Genome Institute under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] 2022. *Docker: Accelerated, Containerized Application Development.* https://www.docker.com/
[2] 2023. *Cloduwatch Agent repository.* https://github.com/aws/amazon-cloudwatch-agent
[3] 2023. *DESeq2.* https://bioconductor.org/packages/release/bioc/html/DESeq2.html
[4] 2023. *DockerHub website.* https://hub.docker.com/
[5] 2023. LangChain. https://github.com/langchain-ai/langchain. Accessed: 2023-08-16.
[6] 2023. *Litophs repository.* https://github.com/lithops-cloud/lithops
[7] 2023. *National Center for Biotechnology Information website.* https://www.ncbi.nlm.nih.gov/
[8] 2023. *Salmon repository.* https://github.com/COMBINE-lab/salmon
[9] 2023. *SRA-Toolkit repository.* https://github.com/ncbi/sra-tools
[10] 2023. *STAR repository.* https://github.com/alexdobin/STAR
[11] 2023. *Terraform website.* https://www.terraform.io/
[12] 2023. *Transcriptomics Atlas repository.* https://github.com/SanoScience/NearData/tree/WORKS23_abstract
[13] 2023. Workflow Description Language (WDL). https://github.com/openwdl/wdl. Accessed: 2023-08-16.
[14] Khairul Alam and Banani Roy. 2022. Challenges of Provenance in Scientific Workflow Management Systems. In *2022 IEEE/ACM Workshop on Workflows in Support of Large-Scale Science (WORKS).* IEEE.
[15] A. Alsaadi, D.H. Ahn, Y. Babuji, K. Chard, J. Corbett, M. Hategan, S. Herbein, S. Jha, D. Laney, A. Merzky, T. Munson, M. Salim, M. Titov, M. Turilli, T.D. Uram, and J.M. Wozniak. 2021. ExaWorks: Workflows for Exascale. In *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS).* 50–57. https://doi.org/10.1109/WORKS54523.2021.00012
[16] J. Alvarez-Jarreta, G. de Miguel Casado, and E. Mayordomo. 2014. PhyloFlow: A fully customizable and automatic workflow for phylogenetic reconstruction. In *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM).* 1–7. https://doi.org/10.1109/BIBM.2014.6999303
[17] Y. Babuji, A. Woodard, Z. Li, D. Katz, B. Clifford, R. Kumar L., Lacinski, R. Chard, J. Wozniak, I. Foster, M. Wilde, and K. Chard. 2019. Parsl: Pervasive Parallel Programming in Python. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* (Phoenix, AZ, USA) *(HPDC '19).* Association for Computing Machinery, New York, NY, USA, 25–36. https://doi.org/10.1145/3307681.3325400
[18] Jonathan Bader, Fabian Lehmann, Lauritz Thamsen, Ulf Leser, and Odej Kao. 2024. Lotaru: Locally predicting workflow task runtimes for resource management on heterogeneous infrastructures. *Future Generation Computer Systems* 150 (2024).
[19] Jonathan Bader, Lauritz Thamsen, Svetlana Kulagina, Jonathan Will, Henning Meyerhenke, and Odej Kao. 2021. Tarema: Adaptive Resource Allocation for Scalable Scientific Workflows in Heterogeneous Clusters. In *2021 IEEE International Conference on Big Data (Big Data).*
[20] Jonathan Bader, Joel Witzke, Soeren Becker, Ansgar Lößer, Fabian Lehmann, Leon Doehler, Anh Duc Vu, and Odej Kao. 2022. Towards Advanced Monitoring for Scientific Workflows. In *2022 IEEE International Conference on Big Data (Big Data).*
[21] V. Balasubramanian, A. Treikalis, O. Weidner, and S. Jha. 2016. Ensemble Toolkit: Scalable and Flexible Execution of Ensembles of Tasks. In *2016 45th International Conference on Parallel Processing (ICPP).* 458–463. https://doi.org/10.1109/ICPP.2016.59
[22] R.A. Carson, S.R. Wopschall, and J.A. Bramwell. 2019. ExaConstit. [Computer Software]. https://doi.org/10.11578/dc.20191024.2
[23] J. Coleman, K. Kincaid, G.L. Knapp, B. Stump, and A.J. Plotkowski. 2023. *ORNL/AdditiveFOAM: Release 1.0.* https://doi.org/10.5281/zenodo.8034098
[24] Sérgio Manuel Serra da Cruz, Maria Luiza M. Campos, and Marta Mattoso. 2009. Towards a Taxonomy of Provenance in Scientific Workflow Management Systems. In *2009 Congress on Services - I.*
[25] Susan B. Davidson and Juliana Freire. 2008. Provenance and Scientific Workflows: Challenges and Opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08).* Association for Computing Machinery, New York, NY, USA.
[26] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. 2017. Nextflow enables reproducible computational workflows. *Nature Biotechnology* 35, 4 (2017).
[27] Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. 2013. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics* 29, 1 (2013), 15–21.
[28] A. Duque and A. Syed. 2023. Phyloflow-Parsl Implementation. https://github.com/grimloc-aduque/Phyloflow-Parsl-Implementation. Accessed: 2023-08-16.
[29] Radmila Hrdlickova, Masoud Toloue, and Bin Tian. 2017. RNA-Seq methods for transcriptome analysis. *Wiley Interdisciplinary Reviews: RNA* 8, 1 (2017), e1364.
[30] G.L. Knapp, J. Coleman, M. Rolchigo, M. Stoyanov, and A. Plotkowski. 2023. Calibrating uncertain parameters in melt pool simulations of additive manufacturing. *Computational Materials Science* 218 (2023), 111904. https://doi.org/10.1016/j.commatsci.2022.111904
[31] Fabian Lehmann, Jonathan Bader, Friedrich Tschirpke, Lauritz Thamsen, and Ulf Leser. 2023. How Workflow Engines Should Talk to Resource Managers: A Proposal for a Common Workflow Scheduling Interface. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid).* Bangalore, India.
[32] Fabian Lehmann, David Frantz, Sören Becker, Ulf Leser, and Patrick Hostert. 2021. FORCE on Nextflow: Scalable Analysis of Earth Observation Data on Commodity Clusters. In *Proceedings of the CIKM 2021 Workshops (CEUR Workshop Proceedings, Vol. 3052),* Gao Cong and Maya Ramanath (Eds.).
[33] A. Merzky, M. Turilli, M. Titov, A. Alsaadi, and S. Jha. 2022. Design and Performance Characterization of RADICAL-Pilot on Leadership-Class Platforms. *IEEE Transactions on Parallel & Distributed Systems* 33, 04 (apr 2022), 818–829. https://doi.org/10.1109/TPDS.2021.3105994
[34] Paul Muir, Shantao Li, Shaoke Lou, Daifeng Wang, Daniel J. Spakowicz, Leonidas Salichos, Jing Zhang, George M. Weinstock, Farren Isaacs, Joel Rozowsky, and Mark Gerstein. 2016. The Real Cost of Sequencing: Scaling Computation to Keep Pace with Data Generation. *Genome Biology* 17, 1 (2016).
[35] OpenAI. 2022. ChatGPT. https://chat.openai.com/chat. Accessed: 2023-08-16.
[36] OpenAI. 2023. ChatGPT plugins: Code Interpreter. https://openai.com/blog/chatgpt-plugins#code-interpreter. Accessed: 2023-08-16.
[37] OpenAI. 2023. GPT Models: Function Calling. https://platform.openai.com/docs/guides/gpt/function-calling. Accessed: 2023-08-16.
[38] M. Rolchigo, S.T. Reeve, B. Stump, G.L. Knapp, J. Coleman, A. Plotkowski, and J. Belak. 2022. ExaCA: A performance portable exascale cellular automata application for alloy solidification modeling. *Computational Materials Science* 214 (2022), 111692. https://doi.org/10.1016/j.commatsci.2022.111692
[39] Josep Sampe, Marc Sanchez-Artigas, Gil Vernik, Ido Yehekzel, and Pedro Garcia-Lopez. 2021. Outsourcing Data Processing Jobs with Lithops. *IEEE Transactions on Cloud Computing* (2021), 1–1. https://doi.org/10.1109/TCC.2021.3129000
[40] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761 [cs.CL]
[41] M. Stoyanov, D. Lebrun-Grandie, J. Burkardt, and D. Munster. 2013. Tasmanian. [Computer Software]. https://doi.org/10.11578/dc.20171025.on.1087
[42] The ExaAM Project. 2023. GitHub UQ. https://github.com/ExascaleAM/Workflows/
[43] The ExaWorks Project. 2023. Software Development Kit. https://exaworks.org/sdk
[44] Paolo Di Tommaso. 2022. *A quick overview of Nextflow workflow system.* https://workflows.community/stories/2022/09/28/nextflow/ accessed 18-October-2022.
[45] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems* 13, 3 (2002).
[46] M. Turilli, A. Merzky, V. Balasubramanian, and S. Jha. 2018. Building Blocks for Workflow System Middleware. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID).* 348–349. https://doi.org/10.1109/CCGRID.2018.00051
[47] J.A. Turner, J. Belak, N. Barton, M. Bement, N. Carlson, R. Carson, S. DeWitt, J.-L. Fattebert, N. Hodge, Z. Jibben, W. King, L. Levine, C. Newman, A. Plotkowski, B. Radhakrishnan, S.T. Reeve, M. Rolchigo, A. Sabau, S. Slattery, and B. Stump. 2022. ExaAM: Metal additive manufacturing simulation at the fidelity of the microstructure. *The International Journal of High Performance Computing Applications* 36, 1 (2022), 13–39. https://doi.org/10.1177/10943420211042558
[48] Kate Voss, Geraldine Van der Auwera, and Jeff Gentry. 2017. Full-stack genomics pipelining with GATK4 + WDL + Cromwell. *F1000Research* 6 (Aug. 2017).