To the Graduate Council:
I am submitting herewith a dissertation written by Paula Olaya entitled "Enabling Reproducibility, Scalability, and Orchestration of Scientific Workflows in HPC and Cloud-Converged Infrastructure." I have examined the final paper copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Michela Taufer, Major Professor

We have read this dissertation
and recommend its acceptance:

Michael Jantz

Jian Huang

Rodrigo Vargas

Yoonho Park

Jay Lofstead

Accepted for the Council:

Dixie Thompson

Vice Provost and Dean of the Graduate School

To the Graduate Council:
I am submitting herewith a dissertation written by Paula Olaya entitled "Enabling Reproducibility, Scalability, and Orchestration of Scientific Workflows in HPC and Cloud-Converged Infrastructure." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Michela Taufer, Major Professor

We have read this dissertation
and recommend its acceptance:

Michael Jantz

Jian Huang

Rodrigo Vargas

Yoonho Park

Jay Lofstead

Accepted for the Council:

Dixie Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Enabling Reproducibility, Scalability, and Orchestration of Scientific Workflows in HPC and Cloud-Converged Infrastructure

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Paula Olaya

August 2024

To my family, friends, and everyone that got me through this journey.

Para mi familia, amigos, y todos aquellos que me acompañaron en este camino.

Acknowledgements

This work is the result of many years of effort and the support and guidance of many people to whom I am extremely grateful for. I would like to express my deepest appreciation to my advisor Dr. Michela Taufer whose guidance and encouragement have allowed me to embrace challenges and grow. She is the personification of many opportunities along my journey that have carved my professional and personal paths in ways that I would have never imagined.

I would also like to thank my mentor Dr. Jay Lofstead whose invaluable guidance, support, advice, and brilliant ideas allowed me to meet the goals of this work.

Massive thanks to my other committee members: Dr. Michael Jantz, Dr. Jian Huang, Dr. Rodrigo Vargas, and Dr. Yoonho Park for their time and precise feedback over the time between my proposal and dissertation.

Words cannot express my gratitude to Sophia Wen and the IBM Research team, including Drs. Talia Gershon, Yoonho Park, I-Hsin Chung, and Seetharami Seelam, for guiding and trusting my work. IBM's support has been fundamental at each development step of our solutions. Many thanks to Ron Brightwell and Sandia National Laboratories for trusting and supporting my work throughout the years.

I had the pleasure of collaborating with different domain scientists who enlightened me with their expertise; Drs. Ricardo Llamas and Rodrigo Vargas whose brilliance was fundamental throughout the development of SOMOSPIE and Drs. Florence Tama and Osamu Miyashita who let me use their fun protein diffraction patterns to create our XPSI framework. Thanks should also go to Giorgio Scorzelli and the

NSDF team who allowed me to broaden my experience from working with materials scientists to developing new software for the project.

I wish to express my gratitude to each member of GCLab (former and current), for their collaboration, conversation, and friendship. Especially, to Drs. Silvina Caino-Lores, Michael Wyatt, Dylan Chapp, Jakob Luettgau, and Jack Marquez whom I admire and consider role models. I would be remiss in not mentioning soon-to-be Dr. Nigel Tan who was my partner in crime throughout this Ph.D. I feel so lucky to know that being part of GCLab has not only given me a Ph.D. but amazing friends that I will treasure forever. Dr. Lauren Whitnah deserves a special mention for her constant direction at the end of this process and for reviewing this document a thousand times.

I could not have undertaken this journey without the support of my family and friends. My parents, Armando and Nancy whose unconditional love always inspires and pushes me to pursue the impossible.

Finally, I thank my fiance, Andrew, for always believing in me and encouraging me to keep going. I would have not been able to do this without you. This one is for you and my parents!

"Be kind, for everyone you meet is fighting a hard battle."

Abstract

Scientific communities across different domains increasingly run complex workflows for their scientific discovery. Scientists require that these workflows ensure robustness; where workflows must be reproducible, scale in performance; and exhibit trustworthiness in terms of the computational techniques, infrastructures, and people. However, as scientists leverage advanced techniques (big data analytics, AI, and ML) and infrastructure (HPC and cloud), their workflows grow in complexity, leading to new challenges in scientific computing; hindering robustness.

In this dissertation, we address the needs of diverse scientific communities across different fields to identify three main challenges that hinder the robustness of workflows: (i) lack of traceability, explainability, and reproducibility; (ii) hidden intermediate data reducing scalability; and (iii) inefficient data management in workflow orchestration. We codesign scientific workflows and HPC and cloud-converged infrastructure to develop robust science, bridging the gap between computational and domain scientists.

First, we develop fine-grained containerized environments that enable data traceability and results explainability by automatically annotating provenance information, to advance widespread reproducibility. Second, we integrate the workflows in HPC and cloud infrastructure and tune the storage technology to enable better I/O and data scalability. Finally, we provide a software architecture that enables efficient data management (scalable and trustworthy data) in the orchestration of scientific workflows while leveraging the high throughput and low latency of node-local storage.

Table of Contents

List of Tables

List of Figures

Chapter 1

Dissertation Overview

1.1 Problem, Motivation, and Proposed Solution

Scientific workflows have become essential for accelerating scientific discovery across different fields, such as earth sciences [1, 2], biology [3, 4], materials sciences [5], and astronomy [6]. They consist of computational applications that transform data to target a scientific quest. These computational applications can leverage big data analytics, artificial intelligence (AI), and machine learning (ML) techniques to automate and improve scientific processes, for example, predicting earth science variables [1], simulating climate change [7], or accelerating drug discovery [3]. Because of these advanced computational techniques, scientific workflows need to run on distributed and high-performance infrastructure that enables efficient data management (high I/O performance, scalable, and trustworthy data). However, as scientists leverage advanced techniques and infrastructure, their workflows grow in complexity, leading to new challenges in scientific computing.

The U.S. Department of Energy's Advanced Scientific Computing Research program in [8] and the scientific workflows community in [9] describe how scientific workflows require start-to-finish collaboration between computational scientists and domain scientists to meet the new and more diverse requirements and challenges

of scientific computing. Additionally, they highlight the importance of developing converged computing infrastructures that must account for new deployment scenarios at the edge and in the cloud alongside traditional HPC data centers. The codesign of scientific workflows and converged infrastructure ensures robust science. Robust science [10] is defined as the capacity for workflows to assure reproducible or confirmable research; scale in performance; and exhibit trustworthiness in terms of the technology, infrastructures, and people.

In this dissertation, we address the needs of diverse scientific communities across different fields (earth sciences [1, 2], biology [3, 4], materials sciences [5], and astronomy [6]) to (i) identify common challenges in terms of reproducibility, scalability, and orchestration and (ii) codesign scientific workflows and converged infrastructure to develop robust science, bridging the gap between computational and domain scientists.

Challenge 1: Lack of traceability, explainability, and reproducibility

Scientific workflows comprise multiple interactive applications that generate, preprocess, model, and analyze large datasets. These interoperable components, including the applications and data, are hard to trace. Additionally, with the rapid growth of AI, most applications integrate ML models, yet these methods have limited transparency and lack explainability. When workflow components cannot be traced and the results cannot be explained, it impedes the workflow's reproducibility.

Challenge 2: Hidden intermediate data reducing scalability

Scientific workflows transform large datasets as different applications are executed. Data-driven or AI/ML workflows produce significant intermediate data that another application reuses, resulting in smaller output data products. Such multistage workflows obscure the complexity of large intermediate data; their underlying computational infrastructure can negatively affect their performance scalability.

Challenge 3: Inefficient data management in workflow orchestration

Scientific workflows run on distributed and heterogeneous infrastructure with data and computation requirements that make their orchestration difficult. A key

aspect when orchestrating workflows is that the allocated infrastructure must ensure efficient data management. This means that the orchestration must ensure I/O performance and scalability while enabling scientists to trust the data for further scientific discovery.

1.1.1 Enabling Traceability, Explainability, and Reproducibility

Scientists deploy workflows to study scientific phenomena; trusting data, methods, software, and hardware becomes more necessary than ever [9]. Trust is important because these workflows are driving scientific decisions that impact society. Scientists can trust their findings only through in-depth data lineage and the complete record trail of the methods generating the results. The data lineage and record trail enable scientists to trace data back to its sources and explain computational methods and their output, to advance widespread reproducibility.

We design a computational environment seamlessly integrated with container technology that automatically creates an in-depth data lineage and workflow execution record trail. To this end, we decouple and encapsulate data and applications of traditionally tightly coupled workflows into individual fine-grained containers. We augment data and application containers to expose provenance metadata and move data across the containerized workflow effectively. Additionally, we create an interface for visualizing and studying the metadata that scientists can use to understand data lineage and computational methods.

Our environment enables scientists to link differences in scientific results back to different input data sources (data lineage for traceability) and link different results to specific methods used, without necessarily mastering all the aspects of implementation and execution of the workflow (record trail for explainability). Given the extra containerization layer, we demonstrate how our environment has limited overhead compared to running the workflows without containers (native environment)

and effectively establishes trustworthiness in an exemplary ML-based earth science workflow, SOMOSPIE [1].

1.1.2 Enabling Scalability

The I/O bandwidth of writing and reading large data contributes to a long makespan of scientific workflows [11]. The infrastructure can negatively impact the I/O performance because different storage technologies and how they are connected to the computational resources result in different I/O performance. Scientists need diverse infrastructures that are properly tuned for the performance requirements of scientific workflows to enable scalability.

We present a methodology for composing workflows on two infrastructures as a service: HPC as a service and a cloud-native service. We contribute insights to the scientists about (i) deployment services for composing their workflows, (ii) technology storage solutions to support large data transformations, and (iii) automatic scalability challenges for complex scientific workflows with the two infrastructures. We describe the architecture layers and how they need to be tuned for the I/O scalability of scientific workflows.

We provide best practices on two infrastructures as a service to enable the scalability of scientific workflows, increasing the productivity of scientists, and accelerating scientific discovery. We demonstrate how different infrastructures impact the performance and how tuning I/O parameters enables the desired scalability. Specifically, we demonstrate the high scalability of the HPC and cloud-converged infrastructure for SOMOSPIE.

1.1.3 Enabling Orchestration

Scientists need to orchestrate their workflows allocating computational and storage resources that ensure efficient data management [12]. Deploying node-local storage technology improves the I/O performance, however, the node-local storage is intended

to be deployed by a single task locked to a single node, while scientific workflows include multiple tasks deployed across multi-node clusters. Scientists need software tools that facilitate the orchestration of these workflows leveraging node-local storage.

We enable scientific workflows to leverage the high throughput and low latency of node-local storage in the cloud while ensuring (i) the persistence of all input, intermediate, and output data artifacts for traceability and trustworthiness and (ii) the shareability of data across tasks and nodes. Our comprehensive approach addresses both the application and system levels in converged infrastructure. At the application level, we enable scientists to automate the workflow's orchestration without any original code instrumentation. At the system level, we manage the node-local storage to provide high I/O performance while ensuring data persistence and shareability.

We demonstrate how our architecture improves the performance and scalability of workflows, enabling more efficient data management practices while reducing I/O overhead and optimizing data flow between computation and storage tiers within the converged infrastructure. We use a benchmark workflow to validate the generality of our solution and a submodule of SOMOSPIE to evaluate its scalability. We demonstrate how our architecture improves the workflow performance (I/O, overall execution time, and scalability) for the two testing workflows.

1.2 Dissertation Statement

We claim that to accelerate scientific discovery while ensuring robust science, domain and computational scientists need to codesign scientific workflows and HPC and cloud-converged infrastructure to (i) support the traceability, explainability, and reproducibility of data and applications, (ii) enable the scalable deployment of scientific workflows, and (iii) facilitate the orchestration and efficient data management of scientific workflows. To validate this statement:

- We develop a fine-grained containerized environment that enables data traceability and results explainability when executing scientific workflows in HPC and cloud infrastructure to advance widespread reproducibility;
- We define an approach to composing scientific workflows in HPC and cloud infrastructure, focusing on I/O and data scalability; and
- We design a software architecture to provide efficient data management during the end-to-end orchestration of scientific workflows in HPC and cloud infrastructure.

1.3 Dissertation Contributions

In this dissertation, we design converged infrastructure to develop robust science that bridges the gap between computational and domain scientists to enable (i) traceability, explainability, and reproducibility, (ii) HPC and cloud-converged and scalable infrastructure, and (iii) orchestration with efficient data management for scientific workflows.

We contribute to tracing, explaining, and reproducing scientific workflows. We design an environment based on fine-grained containerization of data and applications that automatically creates data lineage and record trail of workflow executions (Section 3.3). We implement our fine-grained containerized environment using Apptainer which automatically annotates each container with its data lineage and record trail and effectively supports zero-copy data movement across containers (Section 3.4). We demonstrate the environment capabilities to trace data sources and explain ML results for our ML-based earth science workflow called SOMOSPIE [1] (Section 3.5) and the performance analysis (Section 3.6).

With our design, implementation, and demonstration on a real-scientific workflow of our containerized environment, (i) we identify each data component and attach metadata, enabling traceability of data, (ii) we link results to the methods enabling

explainability of results, and (iii) encapsulate all workflow component and create an execution record trail enabling reproducibility.

We contribute to designing HPC and cloud-converged infrastructure that scales for scientific workflows. We identify the data and scalability complexity of scientific workflows (Section 4.3) and propose a methodology to compose scientific workflows in HPC and cloud infrastructures ensuring I/O scalability (Section 4.4). We implement the methodology to two services: HPC as a service with IBM Spectrum LSF and cloud-native with Kubernetes and tune both infrastructures for I/O scalability (Section 4.4). We perform scalability studies for SOMOSPIE and provide a cost model based on empirical observations for both HPC and cloud services (Section 4.5).

With our design, implementation, and scalability studies on a real-scientific workflow of HPC and cloud-converged infrastructures, (i) we reveal the architecture layers between two HPC and cloud services and integrate common technologies between both, facilitating infrastructure convergence, and (ii) we tune I/O performance parameters in the data architecture layers enabling I/O performance and scalability.

We contribute to the orchestration and efficient data management of scientific workflows leveraging node-local storage. We design a software architecture that guarantees data persistence and shareability when deploying node-local storage on converged infrastructure while ensuring high throughput and low latency for scientific workflows (Section 5.5). We implement our software architecture through Kubernetes/OpenShift operators that act as an HPC burst buffer to make data persistent and distributed file systems to share data across nodes (Section 5.5). We demonstrate the performance of our software architecture for scientific workflows, measuring the generality using a benchmark workflow and the scalability for a domain-specific submodule of SOMOSPIE. (Section 5.6)

With our design, implementation, and demonstration on a real-scientific workflow of our software architecture, (i) we enable workflow orchestration by coordinating the workflow steps and infrastructure without code instrumentation, (ii) we track

data allocated in node-local storage to make it persistent enabling efficient data management (I/O performance and trustworthiness).

1.4 Organization

We codesign scientific workflows and converged infrastructure to develop robust science that enables (i) traceability, explainability, and reproducibility, (ii) HPC and cloud-converged and scalable infrastructure, and (iii) orchestration with efficient data management for scientific workflows. To this end, we organized our dissertation as follows. In Chapter 2 we describe the challenges for scientific workflows and illustrate them using our ML-based earth science workflow, SOMOSPIE; In Chapter 3, we develop our containerized environment to automatically annotate scientific workflows for traceability, explainability, and reproducibility; Chapter 4 defines a methodology for tuning HPC and cloud infrastructure to ensure I/O scalability of scientific workflows; and Chapter 5 describes our software architecture to orchestrate and ensure efficient data management, leveraging the high performance of node-local storage. Chapter 6 provides the findings of this Dissertation along with a description of future work.

Chapter 2

Challenges in Scientific Workflows and the SOMOSPIE Use Case

In this chapter, we define scientific workflows and the different types depending on the data transformation. We identify three common challenges in scientific workflows across different fields in terms of reproducibility, scalability, and orchestration. We present an exemplary ML-based earth science workflow, called SOMOSPIE, to illustrate the three challenges in a real workflow and our proposed solution for each problem.

2.1 Challenges in Scientific Workflows

Scientific workflows have become essential for accelerating scientific discovery across different fields. They allow scientists to compose complex applications by combining heterogeneous codes; defining parameters; managing and generating input, intermediate, and output data; and controlling dependencies. A scientific workflow consists of one or multiple interoperable applications with their software stack and data (i.e., input, intermediate, and output) that scientists can compose and run to study a scientific problem in a well-defined execution environment. We present an abstraction

of a general workflow in Figure 1 with two applications with input, intermediate, and output data.

These workflows use applications (i.e., processing, modeling, analyzing, visualizing) to transform the data. The structure of a workflow can be modeled as direct acyclic graphs (DAGs) where the applications (nodes) are interconnected by data (edges) with a specific scientific objective [13, 14]. The applications in the workflow transform data in four different modalities (Fig. 2): (i) from small input to large output (augmentation), (ii) from large input to small output (reduction), (iii) from large input to large output (reuse), and (iv) from large input to small output with large intermediate data (reduction with large intermediate data artifacts). Multiple data transformations can occur within the same workflow. These workflows deploy advanced computational techniques (i.e., big data analytics, AI, ML, cloud-based services, multi-facility instrument-to-edge-to-HPC) to transform the data that increase complexity [15, 16]. When scientists leverage more advanced techniques and develop more complex data transformations to large data, their workflows grow in complexity, leading to new challenges in scientific computing. We identify three challenges in terms of (i) the lack of traceability, explainability, and reproducibility, (ii) hidden intermediate data and missing scalability, and (iii) inefficient data management in the orchestration of scientific workflows.

2.2 Use Case: ML-based Earth Science Workflow

We use an exemplary ML-based Earth Science Workflow called SOMOSPIE [1] to illustrate the challenges in a real workflow. SOMOSPIE generates, predicts, and analyzes high-resolution topographic data, such as terrain parameters (e.g., slope, aspect, hillshade) and soil moisture. These topographic data are necessary for practical use in earth sciences, including precision forestry and agriculture, hydrology for landscape ecology, and regeneration dynamics [2, 17].

The *terrain parameters (TPs)* such as slope, aspect, and hillshade are topographic data derived from a Digital Elevation Model (DEM) [18]. The parameters comprise meaningful information that describes the surface of the earth. They can be generated at different spatial resolutions and are fundamental in applications such as forestry and agriculture, hydrology, landscape ecology, land-atmosphere interactions, and soil moisture prediction [1,19]. However, generating terrain parameters at high resolution is computationally expensive, hindering their accessibility by the scientific community.

Soil moisture, the percentage of water by weight or volume in soil, is critical for linking climate dynamics to ecosystem functioning, playing a key role in the Earth's water and carbon cycles. The current availability of spatial soil moisture information across large areas (e.g., continents) comes from satellite-based remote sensing sources (e.g., ESA CCI [20], NASA SMAP [21]). However, there are two major limitations of satellite-based soil moisture information: (i) large areas have spatial information gaps (e.g. where there is high canopy density, frozen soil, or extremely dry conditions); and (ii) they have coarse granularity (around 27×27 km grids). There is a pressing need to improve the spatial representation of soil moisture for applications in earth sciences (e.g., ecological niche modeling, carbon monitoring systems, and other Earth system models). Predictions can be used in policy-making and precision agriculture (e.g., optimizing irrigation practices and other land management decisions).

SOMOSPIE is an open-source workflow that consists of four modules: (i) generation of terrain parameters (GEOtiled [22]), (ii) transformation of satellite data and terrain parameters into a format that is suitable for ML modeling, (iii) ML models that transform the satellite data into higher resolution and gap-free predictions using K-Nearest Neighbors (KNN), Random Forest (RF), and Surrogate Based Modeling (SBM); and (iv) visualization methods to rebuild predictions into formats suitable for further study. Figure 3 shows an abstraction of the workflow.

The first module is *data generation, GEOtiled*, which leverages data partitioning to accelerate the computation of high-resolution terrain parameters (aspect, slope, and hillshade) from Digital Elevation Models (DEMs) in a region of interest. The second

module is *data transformation* which combines the terrain parameters with satellite-based soil moisture data to generate the input files for the ML-based prediction module. The third module is *ML-based prediction* which applies traditional ML models to the training and evaluation data from the previous module to predict fine-grained soil moisture for the region of interest. The last module is *analysis and visualization* which takes the soil moisture predictions to do statistical analysis and creation of geographical maps.

2.2.1 Challenges in SOMOSPIE

We describe how SOMOSPIE exhibits the three challenges and our proposed solution for each problem.

Challenge 1: For scientists, the entire SOMOSPIE workflow execution can be opaque, preventing easy data traceability and results explainability. For example, different resolutions of the terrain parameter data (different input data) used for generating fine-grained predictions are not easy to trace from the output. Results obtained using different ML models (different executables) are not easy to link to the specific method used.

In Chapter 3, we present a fine-grained containerized environment that automatically (i) containerizes data and applications and (ii) annotates the workflow execution. Our environment enables scientists to trace and explain the results by linking them to the input data and ML models used during downscaling. Furthermore, the data and applications containers can be reused to generate new workflows without re-writing the application or re-installing the software stack.

Challenge 2: SOMOSPIE transforms large datasets of topographic data (soil moisture and terrain parameters) across the different modules. The data can scale in two directions: The first deals with data that grows because we move from low to high resolutions in a given region (scale up); the second is where data expands as we

cover a larger region of interest (scale out). Scientists require an infrastructure that scales as the intermediate data grows for diverse scenarios.

In Chapter 4, we propose an approach to integrate SOMOSPIE into two cloud services: an HPC service in the cloud (LSF) and an open-source cloud-native (K8s) service with cloud object storage. We tune I/O performance to enable scalability and measure the overall performance when scaling up the resolution (i.e., low to high) and scaling out the region (i.e., from a state to the entire CONUS).

Challenge 3: SOMOSPIE has different modules where each has different data transformations and computational and storage requirements. For example, depending on the region of interest and the resolution for generating TPs and predicting soil moisture, the data grows requiring more computational nodes with larger RAM to execute the workflow. The orchestration of SOMOSPIE considering all the infrastructure requirements and providing efficient data management becomes challenging.

In Chapter 5, we design a software architecture that automates the orchestration of workflows without any original code instrumentation and manages the node-local storage to provide high I/O performance while ensuring data persistence and shareability.

Chapter 3

Tracing, Explaining, and Reproducing Scientific Workflows

In this chapter, we present a computational environment that leverages container technology and provenance annotation to enable traceability, explainability, and reproducibility of scientific workflows. We design an environment that automatically provides an in-depth data lineage and the complete record trail of the methods generating the results. We combine the data lineage and record trail, allowing scientists to trace data back to its sources and explain computational methods and their output, thus advancing widespread reproducibility.

3.1 Problem Statement and Contributions

Scientific workflows play a key role in scientific discovery. These workflows are growing more complex: they consist of different modeling, analysis, and visualization modules; they run on increasingly heterogeneous systems; and they use machine learning (ML) methods with limited transparency. For scientists using these workflows to study scientific phenomena, trusting data, methods, software, and hardware becomes more

necessary than ever [9]. Scientists can trust their findings only through in-depth data lineage and the complete record trail of the methods generating the results.

To enable trustworthiness, we propose a computational environment seamlessly integrated with container technology that automatically creates a workflow execution's record trail and invisibly attaches the trail to the workflow's intermediate and output data. In this chapter, we describe the contributions we make in tracing, explaining, and reproducing scientific workflows:

- We design an environment based on fine-grained containerization of data and applications that automatically creates data lineage and record trail of workflow executions, enabling traceability of data and explainability of results (Section 3.3).

- We implement our fine-grained containerized environment using Apptainer which automatically annotates each container with its data lineage and record trail and effectively supports zero-copy data movement across containers (Section 3.4).

- We demonstrate the environment capabilities to trace data sources and explain ML results for our ML-based earth science workflow called SOMOSPIE [1] (Section 3.5) and the performance analysis (Section 3.6).

We discuss the limitations of current solutions in Section 3.2. We describe the methodology to model fine-grained containerized workflows in Section 3.3 and an implementation of this containerized environment in Section 3.4. Section 3.5 demonstrates the application of our environment to SOMOSPIE for traceability and explainability. We quantify the impact on overhead and performance in Section 3.6. We discuss the interoperability and adaptation of our environment in distributed systems in Section 3.7, and the compendium of functionalities that our solution supports in Section 3.8. Finally, Section 3.9 concludes with a summary of achievements and directions for future work.

3.2 Current Efforts for Traceability, Explainability, and Reproducibility

Annotating a workflow execution with the provenance of its components has been previously used for tracing, explaining, and reproducing results. Related work for collecting and preserving the provenance of a workflow at the system level includes custom file systems tracking provenance such as the Lineage File System (LinFS) [23], PASTA in PASS (Provenance-Aware Storage Systems) [24], and Parrot [25]; encapsulating workflows through ad hoc packages such as CDE [26], ReproZip [27], Umbrella [28], and Occam [29] [30]; and encapsulating workflows through existing container technologies such as Pachyderm [31], REANA [32], and Science Capsule [33]. However, each of these solutions has severe limitations. The use of custom file systems limits portability and usability across systems. Ad hoc packages face the same challenges. Using existing container technology to encapsulate the workflows seems to offer a portable solution for preserving data, applications, and software. However, the current solutions focus on coarse containerization and lack in-depth provenance information.

Containerizing data facilitates transportation, interpretation, and use. We adapt the premise of the data containerization from Data Pallets [34]. It defines storage as a new container type when running workflows, where the containers include the data and links to the application and the input deck. Even when data is containerized, intermediate data is treated as disposable for solutions like Prune [35], CDE [26], ReproZip [27], Umbrella [28], and Occam [29] [30] where they focus on sharing final results of the workflow executions. Only Pachyderm [31] provides a complete audit trail for all data across pipeline stages, including intermediate results. Our solution grants priority access to the intermediate data and its metadata. Moreover, we are the first to permanently and portably attach the provenance invisibly to the data and the applications. We achieve this by using a second partition in the container structure.

Finally, workflow management systems (WMS) have developed and included provenance functionalities in their systems. WMS like Pegasus [12], Kepler [36], and DAGMan [37] provide a way to orchestrate workflows while capturing the data provenance. Only Pegasus provides application containers as a solution to package software with complex dependencies. Pegasus currently supports Docker, Apptainer, and Shifter. However, the workflow managers use the scientific workflow system to track and store the computational steps and their data dependencies but rarely gather information about the environment. Furthermore, integrating a workflow into the management systems can be complex. It requires the translation of the workflow into a specific format: DAX (Directed Acyclic Graph in XML) for Pegasus, XML or KAR files for Kepler, and DAG input file for DAGMan, for example. Finally, not all workflows are at the development stage for managing their pipelines with these workflow tools, which makes a case for alternative solutions for hosting workflows and capturing provenance to allow traceability of data and explainability of results. Our work is intended to perform with these WMS, using their support for containerized workflows to handle the workflow management tasks while we manage the automatic provenance collection and annotate data and applications containers, enabling traceability and explainability.

3.3 Modeling Containerized Workflows

We describe how a scientific workflow can be modeled using our fine-grained containerized environment. We present our solution in terms of the decoupling of workflows into application and data components, the communication between these components, the automatic annotation of each component, and the visualization of the associated metadata.

3.3.1 From Native to Fine-grained Workflow Modeling

Workflows can be executed on native HPC or cloud platforms (Figure 4a). When container technologies are used in HPC or cloud platforms, the whole workflow is usually deployed in a single coarse-grained container (Figure 4b). This coarse-grained containerization enables easy deployment and management. However, the coarse-grained approach makes it difficult to exactly track executions or identify all the workflow components and their interactions for building an in-depth data lineage and record trail. In addition, such an approach does not enable reusability and composability of individual workflow components.

Instead, we decouple the workflow into its components (i.e., applications and data) and use a fine-grained containerized workflow approach that encapsulates each workflow component into its own container (Figure 4c). Each container serves as an immutable object with a unique hash code for permanent identification, enabling easy data lineage and record trail creation. Our approach applies to workflows that can be modeled as DAGs whose nodes (applications) and vertices (data in movement from one application to another) can be both containerized. In theory, any workflow with such features can be abstracted into a fine-grained set of interconnected containers.

3.3.2 Designing Application and Data Containers

Given an application, we containerize it by encapsulating the executable or script with the respective software stack (i.e., OS, libraries, and software packages). Scientists can then reuse the application container across different workflows and reduce the overall storage requirement. Furthermore, in our environment, all application containers are annotated with provenance information including their unique identifier and creation time. The containerization of data is unique to our approach. Because containers are isolated systems, they do not support the persistence of data; consequently, only applications are normally containerized, while the data is hosted in local databases, storage volumes, or images [38–40]. We move away from this implementation by

leveraging the work of Lofstead and co-authors called Data Pallets [34] that defines data as a separate and immutable object once created. To create this object, we define a data container that follows a file-system-in-a-file model. Given an individual dataset (i.e., input, intermediary, or output data), we containerize it into a single and independent data container. Similar to the application container, data containers are augmented to expose provenance information such as the unique hash code, creation time, execution task, and record trail. By doing so, data containers provide trustworthiness, portability, and shareability of their content to users and across workflows. Two important principles inform our fine-grained containerized workflow approach. First, we separate applications and data into their own containers, allowing specificity and unique identification of the components in a workflow for traceability and explainability purposes. Second, we value intermediate data; rather than discharging intermediate data, our approach encapsulates this data to provide a complete preservation of the data lineage for traceability and reusability.

3.3.3 Designing Communication Between Containers

The fine-grained containerization of the workflow components introduces a new challenge in the execution: data has to be moved from one container to the next in an efficient way. When the movement is done through standard container technology, it requires the usage of the host node's storage to serve as a buffer in a two-copy data transfer as shown in Figure 5a. We propose a new communication approach that enables zero-copy data transfer: we bind mount direct paths inside the data and application containers through their namespaces, thus avoiding data sharing via the host. This allows containers to directly exchange data without creating extra copies or using external storage. Ultimately, our approach reduces the time and space needed to transfer data between containers. Figure 5b shows the same example of data transfer for the two-copy approach in Figure 5a but with our zero-copy data

transfer implementation. In essence, an application container can read and write directly from one or multiple data containers.

3.3.4 Designing Annotated Containers

The fine-grained containerization of a workflow allows us to annotate its executions, capturing metadata at a fine-grained level. To deploy the annotation for different workflow executions, the metadata collection has to be automatic. There are three open questions when automatically annotating workflow executions:

1. **Where should the metadata be allocated?** We augment each container with an extra partition within which we allocate the metadata. Our approach enables the tight integration of each workflow component and its metadata, facilitating the traceability of both the individual components and the workflow as a whole. While tightly coupling the metadata partition with the workflow components, our environment effectively stashes the metadata information from regular workflow executions. This means that the data or application itself in the container can never be contaminated by managing the metadata partition. The metadata information can still be accessed at any time through standard container partition access commands.

2. **What metadata should be captured?** We capture the container's identification, creation time, execution task, and record trail. The container identification is a tuple (i.e., `[UUID, name]`) composed of a universally unique identifier (i.e., `UUID`) and a name assigned when the container is created. While the `UUID` is unique, the name can be modified. The creation time captures the point in time when the container is written to disk. The execution task is the set of instructions to run the application. This information includes parameters such as initial conditions, random seeds, and other setting values. The container's record trail is a set of ordered tuples (i.e., `[UUID, name]`)

that defines the pipeline of containers preceding the current one (i.e., data and application containers used before the current container and triggering its use).

3. **How and when should the metadata be collected?** For each container, we expose information embedded in the container environment, which is otherwise hidden from the workflow execution, to collect our metadata. For both data and application containers, the structure of the metadata partition (i.e., `[container's identification, creation time, execution task, and record trail]`) is identical. The content of the partitions depends on the type of container and is collected both statically and dynamically. The static metadata is collected when a container is created and includes the container identification and creation time. The pool of dynamic metadata includes the execution task and record trail. For all containers, (a) the execution task is initialized with `noop` meaning that there is no operation; and (b) the record trail is initialized with `NULL` meaning that there are no predecessor containers because an execution has not happened yet. The dynamic metadata is populated only at the time a workflow is executed. For purposes of reusability across workflows, the static metadata of application containers remains as initialized. On the other hand, the execution task and record trail of the data containers are updated at the execution time to capture the data generation.

Figure 6 shows an example of automatic metadata collection for a workflow with three data containers serving as input ($Input$), intermediate ($Inter$), and output (Out) respectively, and two application containers (App_1 and App_2). The static metadata is initialized with the container identification tuple and container creation time. During the execution of the workflow, the execution task and record trail are updated for the data containers. The figure presents a snapshot of when the whole workflow has finished its execution, meaning that App_1 and App_2 have written the results into the intermediate and output containers respectively. The record trail of the output container includes the output, App_2, and intermediate

containers' identifications. With the identification of the intermediate container, we can retrieve its record trail which includes the intermediate, App_1, and input containers' identifications. We merge the two containers' record trails and define all the containers used in the workflow pipeline. The combination of static and dynamic metadata enables us to build the in-depth data lineage and the complete record trail of the applications generating the results.

3.3.5 User Interface

We provide a user interface to facilitate the study of collected metadata. This interface reads the metadata of one or multiple containers. From this metadata, the interface constructs a workflow visual representation as a directed graph. Scientists can adapt the graph to their needs for further analysis. The nodes of the graph represent each container in the workflow which are labeled by the universally unique identifiers (i.e., `UUID`) and include their respective metadata as their description. Additionally, nodes are distinguished between data and application containers. For each container, the interface backtraces its record trail and connects the nodes in the order of the lineage of that execution. Subsequently, the interface constructs individual subgraphs of each execution. These subgraphs are merged to find common patterns and containers shared across multiple executions, providing a graph with unique nodes (data and application containers) connected based on the data lineage constructed from the descriptions of the metadata.

3.4 Implementing Our Containerized Workflow with Apptainer

We present the design and implementation details of our fine-grained containerized environment. We present the reasons for selecting Apptainer [41] as our container technology to build our environment. We augment Apptainer to create data and

application containers, to enable direct communication during the execution, and to annotate automatically the provenance metadata of the workflow. We develop a Jupyter Notebook [42] interface for metadata visualization and analysis. The code with the augmented Apptainer implementation of our fine-grained containerized environment can be found at: https://github.com/TauferLab/ContainerizedEnv.

3.4.1 Selecting the Container Technology

While there are many different container technologies (e.g., Docker [43], Apptainer [41], Charliecloud [44], Podman [45]), in our work, we use Apptainer for three main reasons. First, Apptainer does not require administrative privileges that are challenging to obtain in tightly controlled environments such as the US National Laboratories. Second, Apptainer's use of the SIF (Singularity Image File) format container allows us to customize the content of each container with different types of partitions (i.e., metadata partition, application partition, data partition), where each container can have one or more partitions. Last, Apptainer supports user-defined add-on functionalities through plugins. These plugins are packages that can be dynamically loaded by Apptainer at runtime, augmenting Apptainer with experimental, non-standard, and vendor-specific functionalities. Some of these functionalities allow users to add commands and flags for the container's creation and execution; the functionalities can serve as an interface with more complex subsystems (i.e., compute and storage devices) at runtime.

We extend Apptainer to feature three functionalities needed to implement the designed fine-grained containerized environment in Sections 3.3.2-3.3.4. First, we use the SIF format container to automatically create the individual application and data containers. We initialize each container with a metadata partition, providing users with access to information otherwise hidden from them. Second, we design a zero-copy data transfer mechanism in Apptainer which facilitates data movement across containers. This zero-copy functionality is now part of the Apptainer code [46].

Last, we automatically annotate the workflow with provenance information. Both automatic creation and annotation functionalities are integrated into a plugin that is part of our software release [47,48]. Furthermore, we implement a Jupyter Notebook that serves as the user interface designed in Section 3.3.5.

3.4.2 Creating Application and Data Containers

We augment the Apptainer runtime with a plugin that supports the automatic creation and execution of fine-grained containerized workflows. The plugin can work with any workflow that can be modeled as a DAG. Given a workflow, our plugin can automatically create a fine-grained sequence of data and application containers using `apptainer workflow --create`. We use a web service on the local machine at port `5000` to facilitate the user with the generation of the fine-grained workflow. Through the web service, the user provides information about the workflow, such as the definition file of an application (i.e., application executable and software stack); the number, location, and size of each input data; and the expected size of the output data. The plugin uses this information to create the workflow with its individual application and data containers. Specifically, the plugin encapsulates individual datasets and application executables or scripts into independent file system partitions. For application containers, the plugin encapsulates the application executable or script together with the software system stack in a squashFS partition. For data containers, the plugin compresses the data in an Ext3 file system partition. Both types of containers include metadata in a JSON generic file system partition.

3.4.3 Zero-copy Communication Between Containers

We extend the Apptainer technology to support direct transfer of data between containers without going through host or external storage (i.e., zero-copy data transfer). We use the bind mount functionality to define a binding path that directly links a directory from the source container to a directory in the destination container.

During execution, the bind path is parsed, capturing which containers and directories to use. The source and destination containers are loaded into the environment. The source directory is replicated in the destination container; any change in the directory inside the destination container is also reflected in the source container and vice versa.

3.4.4 Implementing Annotated Containers

We define a second functionality in our plugin `apptainer workflow --run [workflow_description].json` that grants the user the ability to execute a fine-grained containerized workflow while also annotating containers with metadata. A user executes the fine-grained containerized workflows by running the command `apptainer workflow --run [workflow_description].json`. This command triggers the Apptainer API callback `clicallback` that activates our plugin. Once the plugin is active, it starts the automatic collection of metadata in the workflow. To this end, for each application container's execution, the plugin collects two pieces of information. First, it collects the application's execution task settings (e.g., initial conditions, random seeds, and other setting values). Second, it collects the binding path that lists all input and output data containers for that application container. Following the data containers listed in a binding path, the plugin uses the SIF API to extract each container's identification and creation time. The plugin appends that information to the record trail of output data containers for that bind path. Given an output container, its execution task and record trail are transformed into JSON format and added as a new file descriptor in its metadata partition.

3.4.5 Jupyter Notebook User Interface

We develop a Jupyter Notebook [42] that serves as a user interface for inspecting and gaining insights from the collected metadata. It allows the user to select the metadata of one or more containers and to backtrace the execution dataflow, which is represented through a directed graph. We use the package *NetworkX* expanding the

open-source function *NetworkX Viewer* [49] with a customized node token class to tailor the interactive visualization of the workflow graph. Specifically, our interface enables users to (i) create the nodes based on the list of containers in the record trail where the attributes of each node follow the structure of the metadata; (ii) assign different colors to distinguish between data and application containers; (iii) build independent subgraphs based on the dataflow stated in the metadata of each container; (iv) merge the subgraphs to build larger graphs by using common patterns and unique components; (v) visualize the graphs in an interactive session where the user can reorganize it; and (vi) obtain detailed provenance information about any container.

3.5 Tracing and Explaining Use Case Workflow

Our fined-grained environment is application-agnostic. This means that any scientific workflow that is composed of one or multiple self-contained applications and that can modeled as DAGs whose nodes (applications/tasks) and vertices (data in movement from one task to another) can be containerized. We apply our fine-grained containerized environment to our use case workflow, SOMOSPIE [1] that is composed of multiple self-contained applications for data generation, transformation, ML-modeling, and analysis, and follows a DAG model, as shown in Figure 3. For scientists, the entire workflow execution can be opaque, preventing easy data traceability and results explainability. We demonstrate how our fine-grained containerized approach is applied to two use cases in SOMOSPIE. In the first case, the workflow has to be traced back to the input data to explain different levels of predictions' resolutions (a case of missing data traceability). In the second case, the annotations are used to discover which different ML methods are the reason for different soil moisture predictions when using the same data (a case of missing result explainability).

3.5.1 Integrating Traceability

Figure 7 shows an example of missing data traceability. SOMOSPIE is used with different input data resulting in different levels of detail for the prediction of a region centered around Oklahoma (a rich agricultural area). The longitude is on the x-axis, the latitude is on the y-axis, and each of the pixels/coordinates represents a soil moisture value. Each one of the three figures represents the same area of Oklahoma where the longitude runs from -101.5 to -94.0 and the latitude runs from 33.5 to 37.0. The soil moisture ranges from 0.175 to 0.35 and is mapped into a color gradient where the lowest and driest soil moisture value is red and the highest and most moisturized soil moisture value is blue.

The terrain parameters are selected from different datasets with different resolutions: 1 km, 250 m, and 90 m. The ML method is fed with the input datasets and generates predictions from the satellite resolution (27 km) down to the terrain parameters resolution. However, the figures do not reveal the different resolutions of the input data to the scientists using the same workflow. Our approach annotates the workflow to capture the data provenance in the metadata, providing scientists with full transparency in the data transformations from input to output in the workflow. Figure 8 shows the output of our interface for the predicted soil moisture values in Figure 7. The interface is fed with the metadata of the containers. Based on the metadata, the interface builds and represents the workflow as a graph. The graph shows four data containers (nodes in blue), symbolizing the input data containers connected to an application container (first orange node). This application container corresponds to the ML method that generates the three data containers with the soil moisture predictions at a higher resolution. These predictions are then visualized in the second application container (second orange node), which are encapsulated in three independent output data containers (shown in Figures 7). Based on the graph representation, the scientist can interact with any container and obtain its lineage. Figure 8 presents the metadata partition of each of the containers executed

in the workflow. This metadata partition includes the `UUID` (simplified in the figure), container name, creation time, execution task, and the record trail. The record trail shows the lineage of containers that were used to generate the current component. Starting from the three output data containers (09,10,11) the record trail shows in bold that the same visualization application (08) was executed on three independent predictions' datasets (05, 06, 07). Based on the metadata of the intermediate data containers (05, 06, 07), the record trail reveals (in bold) that each of these predictions was the result of executing the KNN application (04) with the same training data (00) and three independent evaluation datasets with different resolutions (01,02,03). Finally, scientists can trace the three different outputs back to the data sources and explain how the differences come from the three input datasets, each with a different resolution. Furthermore, because our interface directly maps any difference to specific containers used during the execution, it makes it easier for the scientists to retrieve those containers and use them to reproduce the results or generate new studies.

3.5.2 Integrating Explainability

Figure 9 shows an example of a lack of result explainability for the same region. The figure shows three visualizations of the fine-grained soil moisture predictions with the same resolution of 250 m. The three predictions are generated using the same input data but different prediction methods (i.e., KNN, RF, SBM). The figures and associated fine-grained predictions do not reveal any information about the reasoning beyond the sharpness and high mixture of dry and moist values in Figure 9a, the tile-like values in Figure 9b, or the smoother transition between dry to moisturized values in Figure 9c. Once again, our automatic annotation of the workflow generates metadata revealing the differences in the output data, providing the scientists with an explanation of the results in terms of the methodology used. Figure 10 shows the output of our interface for the predicted soil moisture in Figure 9. As in the previous case, the interface is fed with the metadata of the containers, and based on the

metadata it builds the workflow as a graph. The output graph of the interface shows that three possible paths start from two input data containers (both blue nodes). A fork in the execution occurs at the first stage of the workflow where there are three different application containers corresponding to the three different ML methods (first three orange nodes). Each of the ML methods generates an intermediate data container with the soil moisture predictions. Finally, the three predictions are visualized, generating three independent output data containers including the three visualizations (last three blue nodes). We also present the metadata partition for each of the containerized components. By backtracking the record trail, it is possible to reveal that the output containers (10,16,17) were generated by using the soil moisture predictions (06,14,15) from three different ML methods (04,12,13). By providing scientists with the workflow annotations automatically generated by our environment, we enable scientists to explain the different results by linking them to the ML methods used during downscaling. Furthermore, the application containers can be reused to generate new workflows without re-writing the application or re-installing the software stack.

3.6 Measuring Overhead and Performance

We collect the performance measurements of our fine-grained containerized environment and compare them with both a native environment (without containerization) and a coarse-grained containerized environment (for which we containerize all the applications inside a single container and store the data on the native memory). The three environments are part of a broader set of environment configurations with ours (fine-grained) and the native settings at the two ends of the testing spectrum and the coarse-grained environment as a trade-off between the other two. In other words, the coarse-grained environment (single container) serves as the link between the native (no containers) and the fine-grained (multiple independent containers) environments.

3.6.1 Experimentation Platform Settings

We run a diverse set of tests for SOMOSPIE and collect execution time in seconds and storage space in MBs for the different environments. Furthermore, we measure the bandwidth in MB/s for different data read and write workloads. We run the tests on XSEDE Jetstream2 [50] configured as a virtual machine with 16 cores with AMD Milan 7713 CPU type, 60 GB of RAM (DIMM), and 60 GB local disk (HDD) space with an attached volume of 100 GB. In terms of software, the virtual machine has Ubuntu 20.04 as the OS, Apptainer 1.1, Go 1.19, and Python 3.8.10 using the next packages: numpy-1.23.2, pandas-1.4.3, scipy-1.9.0, and scikit-learn-1.1.2.

3.6.2 Execution Times

We measure the execution times of all three environments (i.e., the native, the coarse-grained, and the fine-grained). Specifically, we measure the execution times of SOMOSPIE in Oklahoma with three resolutions: (a) 1 km, (b) 250 m, and (c) 90 m. We run each resolution five times using the three ML methods (i.e., KNN, RF, and SBM) for the different environments on top of Jetstream2.

Figure 11 shows the results. In the figure, we observe that the smaller the data and the more inexpensive the application is, the more time overhead is seen when deploying our fine-grained containerized environment compared to the native (77%) and the coarse-grained (53%) environments. As the data increases and the application becomes more complex and time-consuming, the overhead drops to 10% compared to native and 1.5% compared to coarse-grained. Overhead is always expected given the extra layers of virtualization and the metadata management. Still, as applications are more complex and are deploying larger data, the observed overhead is an acceptable trade-off for the gained traceability of data and explainability of results.

3.6.3 Storage Space

We measure the storage space usage for the two environments at each end of our testing spectrum (i.e., native and fine-grained). The coarse-grained containerized environment matches the data storage space of the native environment since there is no containerization of data, and the application storage space is the same as in the fine-grained. As we encapsulate the workflow components (data and applications) in containers, the containerization adds extra space for the encapsulation format. Depending on the type of container, this extra space is allocated for different purposes. We distinguish between data and application containers and compare the storage space used in the native environment versus our fine-grained containerized environment.

Data containers: In the native environment we have data, and in our fine-grained containerized environment we have data containers. A data container includes the data encapsulated in an Ext3 file system, metadata partition, and the container dependencies. We measure the size of all data containers in SOMOSPIE when executing the three ML methods for the different input data resolutions (1 km, 250 m, 90 m). Figure 12 presents the size comparison between the native environment and our environment for Oklahoma with two resolutions: (a) 1 km and (b) 250 m. On the x-axis, we have the different data (data) and its containerization (c_data): input, c_input, intermediate (inter), c_inter, output, and c_output. Each is represented by a bar indicating their size in MBs, as stated on the y-axis. First, we observe that the container dependencies' space (red bar) is constant. As the data increases in size, the container dependencies' space becomes imperceptible. Second, for all containers, we observe that the space of the metadata partition (green bar) is negligible and always in the order of KBs. Last, we see that the extra space in the data container is mostly part of the Ext3 file system space (orange bar) and is used to encapsulate the data (blue bar) that has the same size in both native and containerized environments. The main cause of the large space used by the Ext3 file system is that it reserves space

for the journaling information and 5% space for root processes. Ways of reducing the reserved space have been explored, and Ext4 [51] incorporates scalability and performance enhancements. However, Apptainer chose Ext3 for their data containers as it is the most efficient and modifiable file system currently available on a wide variety of systems.

Application containers: In the native environment there are applications in the form of scripts or executables that require libraries and dependencies to run. In our fine-grained containerized environment we have application containers that include a script or executable, libraries, a metadata partition, and the container space that corresponds to the OS and software dependencies. Figure 13 presents the comparison between native and fine-grained containerized applications in SOMOSPIE. On the x-axis, we have the three ML methods (application) and their containerization (c_application): KNN, c_KNN, RF, c_RF, SBM, c_SBM, and the visualization (visual) with its containerization (c_visual), all written in Python. On the y-axis, we have the size in MBs. We observe these key properties: first, the scripts and libraries occupy the same space for the native and our containerized environment; second, as in the data containers, the metadata partition is negligible; and last, the extra space in the application container comes from the container space which includes the OS and software dependencies. The identification of libraries, software dependencies, and OS by the application container ensures replicability and transparency of the software system by guaranteeing that the user will always have the same versions of OS, libraries, and software dependencies, regardless of the platform on which the workflow is executed.

3.7 Discussion: Interoperability, Heterogeneity, and Functionalities for Scientists

We discuss our environment's interoperability with other workflows and with resource managers, as well as the adaptation of our environment in distributed heterogeneous systems.

3.7.1 Environment Interoperability

Technology transfer, in which we envision the use of our environment for other applications, has been a key driving factor of our design. Our containerization approach requires that workflows are composed of one or multiple self-contained applications. Furthermore, users should be able to model the workflows as DAGs whose nodes (applications/tasks) and vertices (data in movement from one task to another) can be containerized. Any workflow with such features can be abstracted into a fine-grained set of interconnected containers, making our approach application-agnostic.

We extend the Apptainer runtime to support the concept of automatic creation and execution of fine-grained workflows that can be described as DAGs. We implement the Apptainer plugin described in Section 3.4 which, given a workflow, has the ability to automatically create a fine-grained sequence of data and application containers, as well as the ability to execute these workflows while also annotating containers with execution metadata.

The integration of our environment into resource managers and orchestrators is possible as long as they manage containerized executions. Such containerized executions are supported by HPC and cloud solutions such as Pegasus [12], REANA [32], Pachyderm [31], and Kubernetes [52]. Because data is containerized, the workflow manager does not have to deal with data transfer from to local storage, adding additional portability for our containerized workflow across platforms.

3.7.2 Distributed Heterogeneous Resources

Decomposing the workflow into fine-grained containers enables deploying different components on different nodes in a distributed system as long as the system shares storage (e.g., GPFS, object, or block storage solutions). The bind mount across nodes can work through the shared storage connecting the nodes. For example, when using a cloud platform with an object storage solution, an application container on a node can write to a data container on a different node; the content of the data container can be read by a second application container on the same node or a different node, using Kubernetes as the orchestrator of the containers' execution. With the increase in GPU usage for ML-based scientific workflows, container technologies such as Docker, Apptainer, and Podman support the execution of applications in scientific workflows on GPUs. Specifically, for our containerized environment, Apptainer supports running application containers that use NVIDIA's CUDA GPU computing framework or AMD's ROCm solution. Regardless of the operative system on the host machine, users can run GPU-enabled ML frameworks (e.g., Tensorflow, MXNet, PyTorch). Regarding the data generated by the GPU, we assume that this data is copied back to the CPU, given that GPU-direct is not a widely available technology in most HPC and cloud infrastructures. When data is copied back to the CPU, our fine-grained containerized environment encapsulates it in a data container and automatically collects the data lineage.

3.8 Compendium of Supported Functionalities

In Table 1 we highlight the functionalities of our environment that ensure traceability, explainability, and reproducibility. We can **trace** the execution history of new data and every workflow component, by integrating the data and the metadata, assembling the record trail of data moving within the workflow, and uniquely identifying each component. We ensure **explainability and transparency** by linking the results to

the different methods and algorithms in an easy-to-read record trail, running having a containerized framework, and programmatic metadata access.

We ensure **replicability** and **reproducibility** by automatic collection of workflow metadata, providing a unique identification per each workflow component, and packaging the software system in the application container. We ensure **explainability and transparency** by having a containerized framework, programmatic metadata access, and an easy-to-read record trail.

3.9 Summary of Achievements and Future Work

In this chapter, we present a fine-grained containerized environment using Apptainer technology that enables scientists to achieve trust in findings from their workflows by seamlessly providing data traceability and results explainability. We demonstrate the benefits of our environment for SOMOSPIE, an earth science workflow that uses ML methods to predict satellite soil moisture data to a resolution necessary for policymaking and precision agriculture. Specifically, we use our environment for two use cases in which we trace back differences in predictions due to input data and ML methods. When compared with native and coarse-grained containerized environments, we observe that our environment has limited overhead in terms of time (10%), storage space (5% for data containers and 30% application container), and it has significantly higher IO bandwidth, with a peak of 400 MB/s versus 50 MB/s for native. Our solution is effective for establishing trustworthiness in scientific findings. Future work includes the automatic orchestration of the workflow in our containerized environment and the creation of a catalog of containers that scientists can extend, share, and use to build new scientific workflows in multiple domains.

Chapter 4

Scaling Scientific Workflows in Converged Infrastructure

In this chapter, we define how HPC and cloud can serve workflows with large intermediate data. We present a methodology for composing SOMOSPIE, our exemplary ML-based earth science workflow on two infrastructures as a service: HPC as a service and a cloud-native service. We provide best practices on HPC and cloud infrastructure to enable the shareability and scalability of scientific workflows, increase the productivity of scientists, and accelerate scientific discovery. We demonstrate the scalability of the HPC and cloud infrastructure for SOMOSPIE.

4.1 Problem Statement and Contributions

The underlying computational infrastructure can affect the overall performance of multistage workflows with large intermediate data. The I/O bandwidth of writing and reading large intermediate data contributes to a long makespan of scientific workflows [11]. When scientists compose these workflows for HPC and cloud infrastructure, they must answer critical questions regarding (i) compute orchestration: type and number of compute instances required by the workflow and

the interaction between them; (ii) data management: RAM size, storage technology, and its connection to the compute instances; and (iii) scalability: automatic allocation of resources as the workflow and its data grow.

We define an approach to composing scientific workflows in HPC and cloud infrastructure services, focusing on I/O and data scalability. We describe the complexity of data scalability in workflows, define the technology layers in the HPC and cloud infrastructure, and study the I/O scalability for an ML-based workflow in both infrastructures. In this chapter, we describe the contributions we make toward converging HPC and cloud infrastructure for scientific workflows:

- We identify the data and scalability complexity of scientific workflows (Section 4.3) and propose a methodology to compose scientific workflows in HPC and cloud infrastructures ensuring I/O scalability (Section 4.4).
- We apply the methodology to two services: HPC as a service with IBM Spectrum LSF and cloud-native with Kubernetes and tune both infrastructures for I/O scalability (Section 4.4).
- We perform scalability studies for SOMOSPIE and provide a cost model based on empirical observations for both HPC and cloud services (Section 4.5).

We discuss the related work in Section 4.2. We expand on the complexity of workflows with large intermediate data through our ML-based workflow in Section 4.3. In Section 4.4, we provide the methodology and its implementation of composing workflows in HPC and cloud services, describing the orchestration services, the technology solutions to support the data transformations, and the scalability challenges. We run scalability studies on SOMOSPIE and establish a cost model based on the resulting empirical observations in Section 4.5. In Section 4.6, we discuss how this work is part of an HPC and cloud convergence initiative and the challenges scientists face. We describe how this work supports HPC and cloud convergence and scalability in Section 4.7. Finally, Section 4.8 concludes with a summary of achievements and directions for future work.

4.2 Current Efforts for Scalable Workflows in HPC and Cloud Infrastructure

Several efforts have addressed composing scientific workflows in the cloud that traditionally were deployed in HPC and HTC (high-throughput computing) systems. These workflows are from across domains such as astronomy [53], bioinformatics [54], biology [55], and engineering [56]. In earth sciences, Pangeo [57] prototypes an architecture composed of a cloud object storage and compute cluster. The external cloud storage is dedicated to the easy retrieval of large-volume datasets. Kubernetes [52] (K8s) provisions the compute cluster, and Dask [58] enables computation parallelism. As cloud technology evolves, new services in the cloud (e.g., HPC services and cloud-native services) and storage technologies (e.g., block storage, object storage) emerge to offer elasticity and automation. However, these services are originally tailored for applications far from those in scientific domains. Existing theoretical studies [59, 60] address significant data transformation in the cloud by offering an approach for scientists to explore the cloud for their workflows. However, there are not practical guidelines to tune the I/O, storage, and computational parameters on diverse cloud infrastructures, optimizing the data scalability of the workflows.

In addition to seamlessly connecting and integrating software stacks and data services across cloud platforms for scientific workflows, the virtualization of workflows has become critical for provenance collection [61]. Containerization of scientific workflow enables reusability, portability, and reproducibility of results [48, 62] as well as ease of system maintenance efforts [63–66]. Our work supports these efforts as the containers can be integrated into cloud-native services. As cloud-native services, such as Kubernetes, gain popularity, our work joins other approaches [67, 68] that enable distributed computation of scientific workflows. We emphasize tuning the default settings and parameters for scalability improvements and cost mitigation.

4.3 Data Complexity in Scientific Workflows

We define the data complexity with large intermediate data using SOMOSPIE. We define scalability scenarios that require data decomposition so the workflow can be executed. We focus on the ML-based prediction and visualization modules as the soil moisture prediction values are large intermediate data that bring challenges in terms of resources and performance, resulting in sudden system crashes or unexpected costs. We use the combination of satellite soil moisture and terrain parameters data to train and do inference of an ML model with K-nearest neighbors, random forest, surrogate-based modeling, and other hybrid methods. The generated soil moisture predictions have high resolution (up to 1 m) and are fed into visualization tools to create geographic soil moisture maps and statistics to analyze and extract patterns of soil moisture given time or region.

4.3.1 Scaling Resolutions and Regions

For ML-based workflows, the amount of data is crucial in obtaining better predictions. As data scales, scientists can test the limits of their scientific discovery. We present two ways in which data can scale. The first deals with data that grows because we move from low to high resolutions in a given region (scale up); the second is where data expands as we cover a larger region (scale out). We investigate these scalability scenarios for our SOMOSPIE workflow.

Satellite soil moisture data is collected daily at 0.25 degrees spatial resolution, approximately 27 km [20]. This ESA-CCI database includes records of soil moisture from 1996 until 2020. Scientists input soil moisture and terrain parameter data combinations into SOMOSPIE across time to create different models.

Depending on the target resolution of predictions (i.e., from 27 km satellite soil moisture data to 10 m soil moisture predictions) and the region of interest (i.e., from a state to a continent or worldwide), the intermediate data (i.e., soil moisture points on the map) generated by the ML model may increase exponentially.

Scalability studies using SOMOSPIE or similar workflows in earth sciences target two dimensions: resolution and region scalability. When scaling the resolution of soil moisture, scientists define a region of interest and scale the resolution up to generate finer-grained soil moisture values, resulting in more points of soil moisture values over the same region. For example, Figure 14 shows (a) an example of a satellite resolution of 27 km, (b) a higher resolution of 90 m, and (c) a finest-granularity resolution of 10 m for a region centered around Oklahoma (the Midwest region). We move from 450 to 36 M to 2.9 B data points as we increase the resolution at which we aim to predict soil moisture. When scaling the region, scientists define a resolution and scale out on the map to select a larger area. In Figure 15, we scale from the Midwest region at 10 m resolution with an area of 283,499 km^2 to a much larger region, such as the Contiguous United States (CONUS) at the same resolution with an area of 8,080,464 km^2. In this specific scenario, we move from 2.9 B to 167 B soil moisture data points.

We demonstrate the exponential growth of intermediate data for resolution scalability in the Midwest region when scientists predict soil moisture within the same region but scale from a lower resolution at 90 m with 4.25 GB of total data to a higher resolution of 10 m that increases the total data size to 420.76 GB, increasing the data by 100×. We also document the data growth for the region scalability, where scientists scale from the Midwest region (420.76 GB total data size) to the whole CONUS, expanding the data to 14.93 TB while preserving the same resolution. Table 2 shows three data scenarios for SOMOSPIE that define a region of interest and a resolution at which the scientist aims to predict the soil moisture. Each row represents one of these scenarios: (i) Midwest region at 90 m resolution, (ii) Midwest region at 10 m, and (iii) CONUS at 10 m. For each scenario, we break down the data transformations (i.e., input, intermediate, and output data) and describe the data type, number of points, and the size of each dataset. We observe how input and intermediate data encompass most of that total data size for all scenarios. For all data scenarios, we train our model by using the satellite soil moisture data at 27 km. We average the soil moisture values for the month of January 2010 to ensure the

time scalability factor is constant. The time scalability for the same regions across different months is studied in [19].

4.3.2 Data Partitioning

Scientists apply data partitioning to process temporal and spatial data at different scales. The partitioning enables data parallelism, executing the same application concurrently to different data partitions. Temporal data often exhibit dependencies (e.g., predictions for one year may depend on observations of previous years), making any partitioning along the time dimension often impossible without compromising the accuracy of the ML prediction model. On the other hand, partitioning spatial data is often feasible by considering continuity in a region of interest; scientists can either define a buffer to automatically add information around each partition or use partitions with integrated overlapping neighboring information generated during a pre-processing phase.

We leverage the second type of data partitioning for SOMOSPIE's spatial data. We pre-process the USGS digital elevation models (DEM) to generate terrain parameters of a region of interest that can be partitioned into tiles and deployed for predictions without needing neighbor buffers. Consequently, given a region, we partition it into the number of tiles (tiles_{total}). The tiles are independent of each other, as they already include neighboring information in the terrain parameters and thus can be fed independently into the ML model. Table 3 presents the number of total tiles and the size of the tiles for the Midwest region and CONUS at the two resolutions (i.e., 90 m and 10 m). When we scale up for the Midwest region, we use up to 225 tiles and predict at a higher resolution such as 10 m. The data processed in the execution scale is up to 388.98 GB. Similar data growth occurs when we scale out up to 1,156 tiles, meaning that we go into a larger area, such as CONUS, with the same 10 m resolution, where the workflow encompasses 14.9 TB of input, intermediate, and output data. Figure 16 shows an example of tile distribution for 152 tiles used for

the Midwest region and CONUS at 10 m resolutions in our experiments. Each tile has the same number of points.

4.4 Integrating Workflows in HPC on Cloud

We define a methodology to integrate a scientific workflow with large intermediate data in two different HPC and cloud-native services. We reveal the infrastructure hidden layers for each service and highlight similarities between them. We define the best practices for tuning the infrastructure to obtain better I/O scalability.

4.4.1 HPC on Cloud Services

Traditionally, scientists deploy workflows with large data transformations in HPC systems. However, as these workflows have evolved and the focus has shifted toward scalability, accessibility, and flexibility, the HPC systems show some limitations. For example, the required resources to execute large workflows exceed all resources available in most institutional HPC settings, limiting scientific discovery. To overcome these limitations, we explore two cloud services. These two services aim to bridge the gap between HPC and cloud systems. The first service provides a more traditional HPC approach while the second one was meant for cloud-native services but adapted to support HPC deployments. We can cover two different services and understand their architectural differences, making their usability more transparent to scientists.

Our first service is an HPC service on the cloud. We select the IBM Spectrum load-sharing facility (LSF) cluster close to on-premises HPC but on top of an IaaS platform. It provides a fully automated and configurable deployment of LSF-based HPC clusters in the cloud. Our second service is a cloud-native HPC service using Kubernetes, where we have a container-based platform on top of which we can containerize and execute HPC workflows. Kubernetes provides an orchestration solution for scheduling and automating containerized applications' deployment, management, and

scaling. We execute the same application on both platforms (LSF and Kubernetes). On Kubernetes, the application is containerized. LSF and Kubernetes run on top of a Virtual Private Cloud (VPC) with a secure software-defined network on which scientists can request, configure, and deploy resources on demand. For LSF, virtual machine (VM) instances (nodes) are directly configured inside the VPC. For Kubernetes, Kubernetes runs on all VM instances (nodes) and includes the control plane responsible for managing the Kubernetes objects (e.g., VM instances, pods, PVCs, and PV). A VM instance hosts one or more pods, the most straightforward unit Kubernetes object model. We run one pod per VM instance. LSF has two VM instances: worker and admin (i.e., login instance, LSF management instances, and storage instances). On the other hand, Kubernetes has worker instances only. In LSF, we use a conventional LSF batch system to schedule our jobs; in Kubernetes, we use the standard Kubernetes scheduler. The standard Kubernetes scheduler ensures pods are attached to worker nodes given the resource limitations (i.e., one pod per worker node) but does not support batch scheduling as LSF does.

4.4.2 Data Transformations with Cloud Technology Solutions

In traditional HPC systems, the large intermediate data generated by workflows is typically pushed to parallel file systems or scratch space. When dealing with this data in cloud environments, cloud technology solutions can be used to support the data transformations. We choose the cloud object storage (COS) service, which like a distributed file system, also grants data distribution over multiple instances to provide proportional capacity and throughput scalability. Both LSF and Kubernetes operate with the same COS solution, and we leverage its distributed property to host the input, intermediate, and output data of our workflow into independent COS buckets. There are different packages for mapping object storage into POSIX namespaces, as studied in [69]. We use S3FS [70] for LSF and Kubernetes to map the data in object

storage as a file system. S3FS is a FUSE-based (Filesystem in USErspace, white box) file system that enables the users to read and write data in object storage as if they were from local or HPC file systems. When the application calls the virtual file system (VFS), the VFS invokes the FUSE kernel module that maps the COS bucket data into the VM instance's underlying file system. In LSF, we mount the COS bucket directly to the VM instances through S3FS. In Kubernetes, we use Kubernetes objects, such as a PVC (Persistent Volume Claim) and a PV (Persistent Volume) with an S3FS storage class underneath, to mount the COS bucket into the VM instances. We put all the computation and storage layers together and present the architecture for LSF and Kubernetes with COS for a single VM instance in Figure 17.

HPC on the cloud and cloud-native services come with default settings that scientists often use in good faith, assuming the infrastructure is tuned for their workflow. Unfortunately, these services are tuned for different workflows (i.e., HPC for compute-intensive applications and the cloud for web services). Out-of-the-box settings are not optimal for scientific workflows, but tuning the platform's I/O parameters can optimize scientific workflow performance. We recommend tuning the I/O parameters at a single instance level to ensure performance at a large scale. I/O parameters exposed by S3FS include parallel count, multisize part, and caching. The parallel count refers to the number of concurrent threads requesting the object storage. The multisize part is the size in MB of the chunks transferred from and to the object storage. The cache has two options: location for LSF (i.e., off-instance block storage, RAM) and retention policy for Kubernetes (i.e., auto-cache and kernel-cache).

4.4.3 Scalability in the Cloud

We select cloud services built with the scalability needs of workflows in mind. LSF has a multi-cluster technology and improved job scheduling for enhanced scalability. Kubernetes supports clusters with up to 5,000 nodes, allowing users to scale the full containers used horizontally based on the application requirements. We select COS as

our storage solution because of its flat storing objects approach, which enables scaling to hundreds of petabytes without experiencing degraded performance. We leverage the scalability in our cloud infrastructure configuration to map the parallel data nature of a workflow (Sec. 4.3.2). We map each data partition into an independent VM instance to obtain ideal performance when we have the same number of VM instances as the number of data partitions. Figure 18 illustrates mapping each input data partition to an independent VM instance and writing the intermediate data into a separate COS bucket. We model the trade-off between the number of VM instances vs. the total execution time of the application vs. cost.

We schedule our parallel jobs using the LSF batch system for LSF. However, the standard scheduler in Kubernetes does not support traditional batch scheduling. We therefore schedule the parallel jobs based on a common template, and by using expansions, we define which partition of the data each job processes. As we increase the number of VM instances reading and writing data, we leverage COS's proportional capacity and throughput scalability to host our input, intermediate, and output data. In Figure 18, we present the composition of a workflow in the fourth modality (large intermediate data). We make a representation at the VM instance level connected to three COS buckets for the input, intermediate, and output data, which are common resources for LSF and Kubernetes. We provide insights about the I/O scalability as more VM instances read and write concurrently into independent COS buckets.

4.5 I/O Scaling Use Case Workflow

4.5.1 Tuning I/O Parameters

We integrate SOMOSPIE into two cloud services: an HPC service in the cloud (i.e., IBM Spectrum LSF) and an open-source cloud-native (Kubernetes) service. We study the scalability and the cost of running the SOMOSPIE workflow on the two cloud

services. In particular, we want to understand the effects of scaling up the resolution (i.e., low to high) and scaling out the region (i.e., from a state to the entire CONUS).

We tune I/O parameters for a single tile execution because the tile is the basic data unit we process in each independent VM instance. We define the number of cores in a VM instance and its RAM size based on the SOMOSPIE requirements (i.e., data and application). For the Midwest region at 90 m, we require a RAM size of a minimum of 32 GB, and we select a VM type of 4 cores and 32 GB. The default settings for the S3FS I/O parameters: parallel count is set to 5 threads, multisize part is 10 MB, and caching is off-instance for LSF and auto-cache for Kubernetes respectively. We tune the performance on a single instance using the Midwest region at a 90 m resolution because the region has a single tile for reading (2.43 GB) and writing (1.42 GB), fitting in a single node.

We use FIO (Flexible IO tester) [71] to explore the parametrical space quickly and inexpensively. We run the FIO benchmark for the write operation because it is the most time-consuming I/O operation. We set the FIO jobs to have one sequential write file of 1.4 GB with a block size of 52 MB to match the I/O operation of our application.

We run the benchmark 10 times on a large VM instance with a profile of 48 cores and 192 GB with an Intel Xeon Processor (Skylake, IBRS). We perform an exhaustive search with different combinations of the three S3FS parameters. The parallel count ranges from 5 to 20 threads, the multisize part ranges from 5 MB to 54 MB in size for the transferred chunks to the COS, and the caching considers both a and b for LSF and c and d for Kubernetes. Table 4 presents the write bandwidth obtained by the FIO benchmark when using the default parameters (i.e., five threads, 10 MB chunk size for both services, and caching off-instance for LSF and auto-cache for Kubernetes) and for the tuned S3FS parameters generating the best empirically observed I/O performance (i.e., 12 threads, 10 MB chunk size, and caching in RAM for LSF; 20 threads, 40 MB chunk size, and kernel cache retention policy for Kubernetes).

Based on the results from the FIO benchmark, we make an informed decision regarding the S3FS parameters for executing our SOMOSPIE workflow on the Midwest region at the 90 m resolution on both clusters (i.e., LSF and Kubernetes). We use the same instance type used for the benchmark [i.e., 48 cores and 192 GB with an Intel Xeon Processor (Skylake, IBRS)] to run the workflow. We run our application 10 times, measuring the read and write bandwidth.

Figure 19 presents a boxplot of the measured bandwidth over the 10 runs. We observe an improvement in write bandwidth of 25% for LSF when using the S3FS tuned parameters (Fig. 19a), and 1.3% for Kubernetes (Fig. 19b). As a consequence of tuning the write operation, we also improve the read bandwidth of 17.8% for LSF (Fig. 19c) and 28.4% for Kubernetes (Fig. 19d). With the parameter tuning, we can reduce the read and write bandwidth variability for LSF and Kubernetes and bring the read and write performance closer to each other, closing the gap between platforms. With the distributed nature of our predictions, in which regions are cut in tiles (Sec 4.3.2), we can use the tuned I/O parameters to study predictions at a large scale.

4.5.2 Soil Moisture Predictions at Large Scale

Predicting soil moisture at high resolutions or for large regions requires scalable platforms to enable data growth. Cloud platforms allow scientists to scale resources to support their workflow requirements. However, the scalability consequences in terms of performance for a given platform are often unknown to application teams. We study the I/O performance of the object storage as we scale the soil moisture predictions in resolution (from 90 m to 10 m in the Midwest region) and region (from Midwest to CONUS at 10 m).

We decompose the data into tiles, 225 for Midwest at 10 m and 1,156 for CONUS at 10 m (Table 3). We process each tile in an independent VM instance. We use 2

cores and 16 GB for Midwest at 10 m, and 8 cores and 64 GB VM type for CONUS at 10 m. Figure 20 presents the multi-instances configuration.

As we scale the number of VM instances to match the number of tiles to process them in parallel, we increase the number of I/O requests to the two COS buckets. We measure the I/O performance as we read multiple tiles at the time from the terrain parameters COS bucket and write multiple tiles with the soil moisture predictions to the SM predictions COS bucket. We do a weak scaling test for the object storage solution. In other words, we measure how the read and write bandwidth varies with the number of VM instances processing a tile with the same size per instance.

We present the weak scaling results for the Midwest region at 10 m resolution in Figure 21. We consider up to 225 tiles that read 1.2 GB of data per tile and write 685 MB (Figure 16a). The number of tiles, and the associated number of VM instances, range as follows: 8, 16, 24, 32, 48, 94, 152, and 225 (the entire Midwest region). We apply some resource constraints on the HPC service in the cloud to mimic the scientists having access to on-premise HPC systems (i.e., limited allocations and computational nodes). On LSF, our experiments are constrained to 200 cores that can be split between the worker and admin instances. We use a worker VM instance of 2 cores and 16 GB; by eliminating the cores for the admin VM instances, we scale up to 94 VM instances in LSF. Instead, in the cloud-native service, Kubernetes, we do not have any restrictions in terms of computational resources, so we can scale up to the number of VM instances. For LSF, the COS bucket maintains I/O performance when we write 64.4 G (Figure 21a) and read 112.8 GB (Figure 21c) of data in parallel from 94 nodes for LSF for the Midwest region at 10 m resolution. For Kubernetes, we observe that the I/O performance of the object storage scales as we write 135.9 G (Figure 21b) and read 248 GB (Figure 21d) of data in parallel from 225 nodes. We translate the performance into accumulated bandwidth across all VM instances. For LSF, we reach 864.8 MB/s write bandwidth and 5.6 GB/s read bandwidth, having 94 VM instances. For Kubernetes, we reach 2.4 GB/s write bandwidth and 11.2 GB/s read bandwidth having 225 VM instances. Overall, we observe no I/O performance

degradation in the object storage as we increase the number of instances of reading and writing in parallel for LSF and Kubernetes. We note higher bandwidth variability (Figures 21b and 21d) in the Kubernetes service that is attributed to the additional layers in the cloud-native service architecture and the virtualization of resources.

We increase the tile size to demonstrate I/O performance with a similar number of VM instances (i.e., 8, 16, 24, 32, 48, 94, 152) but a larger problem size (larger tiles) using the CONUS at 10 m scenario. We increase the tile size by 4 and now read tiles of size 5.1 GB and write 2.8 GB soil moisture predictions for each tile. We use a worker VM instance of 8 cores and 64 GB for this scenario. By eliminating the cores for the admin VM instances, we can scale up to 24 VM instances in LSF before hitting the platform's resource constraints. We scale to 152 for the CONUS and stop because of budget allocation limits. We observe that the I/O performance of the object storage scales up when writing 425.6 GB of SM data point in parallel with 152 VM instances in Kubernetes (Figure 22b). For LSF, the object storage I/O performance scales as we write 67.2 GB (Figure 22a) and read 122.4 GB (Figure 22c) of data in parallel from 24 nodes for CONUS at 10 m resolution. The reading in Kubernetes (Figure 22d) scales up to 94 VM instances (479.4 GB); when we reach 152 VM instances (775.2 GB), the performance drops from 80 MB/s to 65 MB/s. This is an empirical trend that can be further investigated in future work. We translate the performance into accumulated bandwidth across all VM instances. For LSF, we reach around 240 MB/s write bandwidth and 1.8 GB/s read bandwidth, having 24 VM instances. For Kubernetes, we reach 1.6 GB/s write bandwidth and 10.6 GB/s read bandwidth having 152 VM instances. We obtain limited scalability performance results when we limit the number of VM instances for LSF, mimicking the limited access to educational resource allocations. Alternatively, we demonstrate scalability with Kubernetes (cloud-native service) as we stretch the resources under stress conditions.

4.5.3 Modeling Costs for Soil Moisture Prediction

As we scale the data in the workflow, the prediction costs grow. When scaling to higher resolutions or larger regions, a workflow like SOMOSPIE allows for saturating all resources available in most institutional settings. The consequence of this limit is that we have to preprocess the tiles in multiple batches one after another. A benefit of cloud environments is that we can assume virtually infinite resources, allowing us to preprocess all tiles in parallel in constant time. We aim to understand the tradeoff between time and cost for all the resources. We develop a model for calculating the costs of predicting soil moisture in LSF and Kubernetes. Our model measures the total cost for the computational, I/O, and storage resources. The storage cost for the object storage technology is $cost_{storage} = \frac{\$USD}{hour} * capacity$, where $capacity$ is the total data stored in the bucket. The storage cost is the same for LSF and Kubernetes.

We establish our computational and I/O cost model with t_{tile} as our basic unit. This is the time in hours that it takes to read, infer high-resolution soil moisture values (computation), and write the predictions for a single tile (Eq. 4.1).

$$t_{tile} = t_{read} + t_{compute} + t_{write} \tag{4.1}$$

We establish the cost of a VM as in Eq. 4.2. Depending on the size of the tile, we define a VM type for our worker instances in terms of the number of cores (N_{cores}) and the RAM capacity (RAM_{size}). Each cloud vendor has a $\frac{\$USD}{hour}$ rate for the CPU and memory in the VM instance.

$$cost_{VM} = (\frac{\$USD_{cpu}}{hour} * N_{cores} + \frac{\$USD_{mem}}{hour} * RAM_{size}) \tag{4.2}$$

When we scale to all the tiles for a region of interest, we consider the number of tiles in the region ($tiles_{total}$) and the number of worker VM instances available in the LSF and Kubernetes clusters ($N_{workerVMs}$). With these two variables, we can calculate the ratio of batches of tiles that will be processed in parallel as $\left\lceil \frac{tiles_{total}}{N_{workerVMs}} \right\rceil$.

We multiply this ratio by the t_{tile} to obtain the total time (t_{total}) that it takes all worker VM instances in the cluster to execute the application. We determine the computational cost once we have the time for all tiles. To this end, we use the cost for the VM type we define for our worker instances ($cost_{workerVM}$) and the number of worker instances available ($N_{workerVMs}$). Combining the time it takes to execute all tiles, the cost of a worker VM, and the number of worker instances, we define the computational cost for all tiles in a region as in Eq. 4.3.

$$\begin{aligned} cost_{compu} =& N_{workerVMs} * cost_{workerVM} \\ & * \left\lceil \frac{tiles_{total}}{N_{workerVMs}} \right\rceil * t_{tile} \end{aligned} \tag{4.3}$$

The computational cost in Kubernetes for all tiles is the same as $K8scost_{total} = cost_{total}$ because the Kubernetes cluster has only worker instances. However, the LSF cluster and the worker instances include the admin instances. Our LSF cluster has one login, two management, and one storage instance. We have the same VM type for all of them and refer to its cost as $cost_{adminVM}$. Therefore, we define the computational cost for LSF as the sum of the worker instances and the LSF admin VMs, as in Eq. 4.4.

$$\begin{aligned} LSFcost_{total} =& (4 * cost_{adminVM} \\ & + N_{workerVMs} * cost_{workerVM}) \\ & * \left\lceil \frac{tiles_{total}}{N_{workerVMs}} \right\rceil * t_{tile} \end{aligned} \tag{4.4}$$

We apply our model to the three data scenarios and present the computational costs for LSF and Kubernetes in Table 5. Each cloud vendor provides a rate for their instances in terms of CPU ($\frac{\$USD_{cpu}}{hour}$) and memory ($\frac{\$USD_{mem}}{hour}$). After checking different cloud vendors rates, we define $\frac{\$USD_{cpu}}{hour} = 8cents/h$ and $\frac{\$USD_{mem}}{hour} = 7cents/h$. For LSF, we establish the same type of VM for our admin VM instances with two cores and 8 GB of RAM. We calculate the t_{tile} by taking the geometric mean across

all tiles and different runs for each stage (i.e., read, compute, and write) and data scenario. We observe that given the worker VM instances in the clusters, the total time processing all the tiles is higher in LSF than in Kubernetes since in Kubernetes we can run all tiles in parallel. A second remark is that the computational cost for Kubernetes is lower than LSF for Midwest at 90m and 10 m. As there are more tiles, as in the CONUS at 10 m example, two factors seem to penalize the cost for Kubernetes, making it higher than for LSF. The first factor is the time per tile because 1156 instances in parallel in the cloud augment the variability and decrease the compute performance. The second factor is the number of VM instances; even though in LSF the 24 worker VM instances would take around 11.8 hours, there is a tax for the amount of worker VM instances in Kubernetes.

4.6 Discussion: HPC and Cloud Convergence and Functionalities for Scientists

We discuss (i) our initiative to bridge the gap between HPC and cloud infrastructures and communities and (ii) the technology transfer limitations that scientists and users experience when composing their workflows in this converged infrastructure.

4.6.1 HPC and Cloud Convergence

Scientific workflows require resources (i.e., CPUs, RAM, disk) that exceed the capacities of local workstations. Traditionally, scientific workflows like Montage [72], LIGO [73], and Epigenomics [74] run on HPC resources. HPC provides dedicated and optimized resources for computational performance. However, they lack portability, elasticity, shareability, and accessibility impacting the scientific community.

Scientists have started moving their workflows to cloud computing infrastructure (e.g., Montage [53], LIGO [75], Epigenomics [76]). Cloud resources provide

configurable, elastic, accessible, and scalable computational environments. However, the performance is tuned for microservices and not scientific workflows.

Our work is part of a larger initiative of HPC and cloud-converged infrastructure to support scientific workflows. The objective is to identify the limitations between converging both worlds, reveal the opaque hardware and software components in both resources, and design tools that provide common and transparent layers between HPC and cloud infrastructure for scientists to develop and execute their workflows.

4.6.2 Technology Transfer Limitations

We identify two limitations when composing scientific workflows in HPC and cloud resources. The first limitation is the obscure nature of the hardware and software architecture layers in both resources. Cloud computing has an enterprise component, therefore each cloud vendor develops their jargon, software, and hardware to offer the most competitive solution. There are some efforts to standardize concepts, technologies (e.g., Kubernetes, OpenShift, OpenStack, etc.), and newer APIs to transfer between cloud vendors and to enable transparency to the user, but this comes with a cost. Compared to cloud resources, HPC centers have more transparency available to the users as these resources are managed by national laboratories or government institutions. However, access to the resources is not always granted, and managing the software stack and all dependencies hinders scientific development. In addition to the impractical necessity of ad hoc re-engineering codes for new hardware to ensure performance as new HPC centers and technology are developed. In the last years, new efficient package managers have been built to automate the installation and fine-tuning of simulations and libraries in supercomputers, and portable ecosystems have been introduced, but this still comes with a performance overhead and these ecosystems conceal important hardware characteristics from developers.

The second limitation is the availability of technologies depending on the data or HPC center, which leads to technology lock-in. For example, in storage technology,

scientists optimize their workflows to deploy data in parallel file systems, local disk, or scratch space in HPC centers. In cloud data centers, scientists deploy file, block, or object storage. Storage classes and drivers are created to enable transferability from cloud technologies to HPC, however, each cloud vendor implements and provides a few, if not only one, of these storage classes and drivers with out-of-the-box performance. This limits scientists and developers who might not be aware of other tools or tuning methods that offer better performance and can be added to their cloud centers. In [69], we provide a characterization and performance testbed of storage packages that allow transferability from cloud object storage to file system. This work enables scientists to understand the available technologies and compare their performance depending on the data transformations in their scientific workflows.

4.7 Compendium of Supported Functionalities

In Table 6 we summarize and highlight the functionalities of the first steps towards developing HPC and cloud-converged infrastructure and ensuring workflow scalability. We work towards HPC and cloud convergence by revealing the architecture layers in both infrastructures, defining common technologies that serve as a bridge between both, and facilitating the infrastructure deployment to scientists. We ensure scalability by designing a distributed infrastructure that is flexible and tuned for the scientific workflow performance.

4.8 Summary of Achievements and Future Work

In this chapter, we study the composition of large-scale scientific workflows that deal with large intermediate data on two services. The first one is LSF, an HPC as a service on cloud resources, close to on-premise HPC but on top of an IaaS platform. The second is Kubernetes, a container-based PaaS platform where workflows are containerized and executed. We provide best practices on cloud infrastructure to

enable the shareability and scalability of scientific workflows, increase the productivity of scientists, and accelerate scientific discovery. We demonstrate the scalability of the cloud infrastructure for SOMOSPIE. SOMOSPIE uses ML models to predict satellite soil moisture data to a resolution necessary for policymaking and precision agriculture. Specifically, we measure performance when scaling up the resolution (i.e., low to high) and scaling out the region (i.e., from a state to the entire CONUS). We reach an accumulated write bandwidth of 864.8 MB/s and 5.6 GB/s accumulated read bandwidth having 94 VM instances in LSF. We obtain 2.4 GB/s write bandwidth and 11.2 GB/s read bandwidth having 225 VM instances in Kubernetes. Future work includes continuing the HPC and cloud convergence initiative by focusing on scheduling policies on Kubernetes and offering operators that facilitate the automatic composition of scientific workflows.

Chapter 5

Orchestrating Scientific Workflows with Persistent and Shared Data

In this chapter, we design a novel software architecture that enables the orchestration and efficient data management (scalable and trustworthy data) of scientific workflows while leveraging the high throughput and low latency of node-local storage in the cloud. We address the challenges of lack of persistence and shareability when deploying node-local storage. We provide a holistic solution including the application and system levels in hybrid cloud environments. We demonstrate how our architecture improves the performance and scalability of GEOtiled [22], a submodule of SOMOSPIE. It enables efficient data management practices while reducing I/O overhead and optimizing data flow between computation and storage tiers within a hybrid cloud environment.

5.1 Problem Statement and Contributions

The common object storage model used in cloud computing was originally designed for services; it was not optimized for complex scientific data transformations that require high throughput and low latency when handling large intermediate data [77].

Node-local storage offers high throughput and low latency capabilities and is available in cloud platforms. However, these node-local SSDs (solid-state drives) are intended to be deployed by a single task locked to a single node, making them unsuitable for scientific workflows that include multiple tasks (applications) deployed across multi-node clusters. This leads to two main problems: (i) lack of persistence because data is ephemeral to the node and (ii) lack of shareability because data is local to the node where the task is executed.

We enable scientific workflows to leverage node-local storage in the cloud while ensuring (i) the persistence of all input, intermediate, and output data artifacts for traceability and trustworthiness and (ii) the shareability of data across tasks and nodes. We implement our solution through a novel software architecture called Persistent, Shared, and Scalable Data (PerSSD) and demonstrate its effectiveness. We use a benchmark workflow to validate the generality of our PerSSD and GEOtiled to evaluate its scalability. In this chapter, we describe the contributions we make in orchestrating scientific workflows while leveraging node-local storage:

- We design PerSSD, a software architecture that guarantees data persistence and shareability when deploying node-local storage on hybrid cloud while ensuring high throughput and low latency for scientific workflows (Section 5.5).
- We implement PerSSD through Kubernetes/OpenShift operators that act as an HPC burst buffer to make data persistent and distributed file systems to share data across nodes (Section 5.5).
- We demonstrate the performance of PerSSD for scientific workflows, measuring the generality using a benchmark workflow and the scalability for a domain-specific workflow, GEOtiled. (Section 5.6)

In Section 5.2, we discuss the limitations of current solutions for workflow orchestration and node-local storage management. Section 5.3 describes the infrastructure used for scientific workflows with large data transformations. We identify

the challenges when using node-local storage in Section 5.4 and propose our novel architecture that addresses the lack of persistence and shareability challenges in Section 5.5. We evaluate the performance of our solution on a benchmark workflow to demonstrate its generality and on an earth science workflow to assess its scalability in Section 5.6. We discuss how schedulers impact this work in Section 5.7. We list our architecture's compendium of supported functionalities: trustworthiness and scalability in Section 5.8. Finally, Section 5.9 concludes with a summary of achievements and directions for future work.

5.2 Current Efforts of Workflows Orchestration with Node-local Storage

Scientific Workflow Orchestration: Using workflow management systems (WMS) is the most traditional approach to orchestrate the execution of diverse workflows on heterogeneous computing resources [78–80]. However, there is a whole variety of complex systems; many WMS make the orchestration of the workflow more difficult and less general and portable, thus hampering its deployment by others. As a solution to the lack of generality and portability, container orchestration has become part of the standard; instead of managing the workflow components directly, each component becomes isolated and standalone units of software that are orchestrated [13, 81, 82]. Many WMS have included container orchestrations as part of their services [83–85]. However, the HPC-based WMS [86–88] are costly for development and maintenance, and the cloud-based solutions [89–91] have a high cost and high complexity level to avoid vendor lock-in. Kubernetes [52] is open source and provides a set of well-standardized principles and practices for running containerized workflows. WMS, such as Pegasus [12], Nextflow [84], Galaxy [92], Airflow [93] have integrated Kubernetes support at different levels.

Our work stands out as we propose a cloud-native solution by expanding the orchestration functionalities of Kubernetes by integrating a pipeline operator. The pipeline operator is open-source so it can be deployed in any public cloud and on hybrid cloud without vendor lock-in. Additionally, it does not require any instrumentation of the original code or pre-knowledge about Kubernetes resources by the domain scientists.

Node-local Management: Node-local storage management has been a widely studied topic in HPC. We observe efforts to automate the configuration of a two-tiered storage architecture depending on the I/O requirements of the scientific workflow [94,95]. Burst buffers have been commonly deployed [96] to combine the two storage tiers and have proven vast I/O performance improvement [97, 98]. However, there are not many efforts that address the same problem in cloud environments. Most work [77, 99, 100] that deploys workflows in hybrid cloud relies on tier 1, external object store technology that depends on the network to perform. Work that explores two-tiered storage in hybrid cloud relies on middleware solutions for the data orchestration [101] or focuses on scheduling techniques [102, 103]. Nonetheless, these approaches do not address data persistence and shareability challenges of node-local storage. Authors in [104] maintain the consistency between the two tiers in hybrid cloud, however, they use the object storage as a cache but the primary storage is tier 1. Cloud-native solutions like OpenEBS [105] and the Fusion file system [106] offer the possibility to set up persistent storage when using node-local SSDs. However, both solutions depend on the vendor storage classes available and have been mostly tested in only two cloud vendors, hindering the portability of the solution. Additionally, the Fusion file system is an enterprise solution (not open source).

Our work is distinctive; we propose a cloud-native solution that directly leverages node-local storage and ensures persistency and shareability in hybrid cloud environments. Our solution takes inspiration from HPC burst buffers to properly manage the node-local data and improve I/O performance. We asynchronously trace the data transformations and push it from ephemeral node-local to persistent external

storage. We additionally create a database keeping track of the workflow components and annotating them with important provenance information.

5.3 Scientific Workflows and Two-tiered Storage Infrastructure

As scientists explore new infrastructure and new algorithmic techniques to optimize their workflows, there are new challenges in the data management and orchestration of those workflows. In this section, we define the data complexity of scientific workflows and stress the importance of the underlying infrastructure to enable the required performance while ensuring the widespread trustworthiness of workflows and the results.

5.3.1 Scientific Workflows and Data Transformations

Data is fundamental in scientific discovery. As data scales, scientists can expand the boundaries of their investigations. Scientists design workflows that transform the data to obtain insights that lead to scientific breakthroughs. Multiple data transformations can occur within the same workflow, as explained in Chapter 2.

Data continues to grow, so the workflows correspondingly increase in complexity. However, even as workflows are complex, performance requirements particularly in terms of scalability remain. In addition to ensuring performance, we observe an increasing need in the community to ensure the trustworthiness of scientific discovery [48,62]. To this end, scientists using workflows to study scientific phenomena need to trust all the components involved during the execution. Making intermediate data accessible and tracing its transformations is key for scientists to build trust in the workflow and understand scientific results.

5.3.2 HPC and Hybrid Cloud Storage Infrastructure

The infrastructure where these scientific workflows and data transformations are deployed plays a critical role in their execution performance and the trustworthiness of the results. Depending on the workflow (i.e., computation-heavy, memory-heavy, or I/O bound), some infrastructures are more suitable than others to address the workflow requirements. For instance, a computation-heavy workflow can run on multiple larger CPU or GPU nodes, while a memory-heavy workflow is best deployed on nodes with larger RAM and disks. In addition, the underlying infrastructure can also introduce variability in the accuracy of the scientific workflows [107, 108].

HPC and hybrid cloud are common infrastructures for scientific workflows, but they have important differences in design that affect performance. Figure 23 presents a diagram comparing HPC and hybrid cloud infrastructure, highlighting the two-tiered storage design. This storage design comprises tier 1 with across-cluster (nodes and users) data accessibility and tier 0 with direct compute-to-storage access. The figure identifies the technology deployed for each infrastructure and an approach to combine both tiers, obtaining better performance and persistence.

HPC infrastructure has been the preferred platform for deploying scientific workflows with large data transformations because of the dedicated, on-premises, and optimized resources for computational and storage performance. Traditionally, large supercomputers operate a parallel file system in their tier 1. Parallel file systems allow multiple users and applications to read and write simultaneously to a single or multiple large files. In tier 0, HPC systems offer node-local SSD storage, which has been widely used because of its significant performance advantages (i.e., low latency and high bandwidth). Even though node-local storage offers low latency and high bandwidth, the data is ephemeral to the node life cycle; when the node stops, fails, or gets rebooted, data is lost. This translates into data persistence and shareability challenges.

In current HPC systems, burst buffers have been deployed to combine the two storage tiers: node-local SSDs and the parallel file system. Burst buffers use node-local SSDs as a buffer between the compute and the end storage tier (parallel file system) to improve the bandwidth. With this approach, tasks in the workflow write data to node-local storage, obtaining the best performance. At the same time, the burst buffer asynchronously moves the data to the parallel file system, making it persistent.

Hybrid cloud infrastructure facilitates elastic and automatic orchestration and deployment of scientific workflows. Hybrid cloud enables scientists to run any scientific workflow consistently across any footprint, including on-premise, at the edge, and in the cloud. In other words, a hybrid cloud infrastructure provides an HPC and cloud-converged infrastructure that scientific workflows can leverage. Given the elasticity of hybrid cloud, there are several storage solutions. As in the case of HPC, we select object storage as our tier 1 storage because it distributes data over multiple instances so users and applications can read or write in parallel. Hybrid cloud also offers node-local SSD storage in tier 0. However, we note that in available infrastructures, the vendors warn users about how data on the node-local SSD needs to be self-managed to ensure persistence and shareability.

5.3.3 Hybrid Cloud Orchestration

Hybrid cloud infrastructure gives scientists control over their resources and customizes those resources based on workflow demands. For example, an academic laboratory with a small dedicated cluster could host their sensitive data while computing more heavy workflow stages in the public cloud. Moreover, the public cloud grants on-demand creation of clusters with customized resources, such as a scalable number of virtual instances with different instance types (i.e., number of cores or with specialized hardware) and storage technology.

The elasticity of hybrid cloud infrastructure is a consequence of the virtualization of the computational layers. For example, in Figure 23, we observe bare-metal nodes for HPC, whereas, in hybrid cloud, the resources can be virtualized (i.e., accessed through virtual machines VMs). Furthermore, container orchestration has become the standard for portable and flexible computation of scientific workflows in hybrid clouds.

Container management systems like Kubernetes and OpenShift enable hybrid cloud, meaning that they enable users to run any workflow consistently across any footprint, including on-premise, at the edge, and in the public or private cloud. In Figure 24, we present the general architecture for a Kubernetes or OpenShift cluster. The control plane manages the cluster with the API server as its core. The API server exposes an HTTP API that lets end users, different parts of your cluster, and external components communicate with one another. The cluster consists of a set of virtual machine (VM) instances running on top of computational nodes. Since we assume that a node can just host a single VM instance, we use node and VM instance interchangeably across the chapter. Each *VM instance* hosts one or more *pods* that are the foundational unit of these container management systems. Each *pod* hosts a workflow task that runs in a *container*.

On-disk files in a *container* are ephemeral; therefore, *volumes* can be used to connect storage to the computational resources. Ephemeral volume types have a lifetime of a pod, but persistent volumes (PV) exist beyond the lifetime of a pod, enabling the traceability and trustworthiness of results. Users can use persistent volume claims (PVCs) to request persistent volumes (PVs) based on the storage type in the cluster, for example, cloud object storage (COS). Another PV is a hostpath that allows mounting a file or directory from the host node's filesystem into the Pod.

When working in hybrid cloud, scientists need tools that facilitate the mapping of the infrastructure (e.g., pods, volumes, etc.) to the workflow components (tasks and data) so it can be executed and tuned for performance.

5.4 Node-Local Challenges in Hybrid Cloud

Node-local storage relieves the bandwidth burden on storage tier 1 by providing an SSD buffer layer between the compute and storage systems, offsetting constant I/O demands. The performance gains from node-local storage require certain trade-offs in the data, specifically in its persistence, durability, and shareability. In other words, the low latency and high bandwidth of node-local storage are a consequence of the direct connection with the computational resources. However, that brings two challenges when executing workflows: data persistence and data shareability. We describe these two challenges in the context of scientific workflows deploying in hybrid cloud infrastructure.

5.4.1 Challenge 1: Data Persistence

The persistence of data is essential to explain and trust scientific workflows. When data is persistent and scientists can access and trace it, they can understand the data transformations within the workflow and, therefore, rely on trustworthiness in the process and the results.

When a workflow deploys node-local storage, the data is ephemeral, meaning that the data has a node life cycle. In hybrid cloud, this is even more challenging because resources are shared by many users, and the system can have failures. Therefore, when other users are allocated to the same physical node, when the node is rebooted, or when there is a node failure, all data is lost. Most public clouds warn the user about their responsibility to move the data from node-local storage to persistent storage. Additionally, even when assuming that the workflow is executed on non-shared and dedicated resources, the accessibility of the data by other users and scientists to replicate or expand on the work can limit the reach of the discovery. Making the data persistent (moving it from node-local to persistent storage) is essential to control the capacity of node-local storage devices that have a more limited available space. Most systems offer node-local SSDs ranging from 100 GB to a maximum and

not very common 10 TB. However, for some scientific workflows [1, 3, 4] when the input, intermediate, and output data are hosted in the node-local storage, the node runs out of space, stopping the execution. We observe the need for software tools that asynchronously make data persistent by moving it from tier 0 to 1 during the workflow's execution while cleaning the available space in tier 0.

5.4.2 Challenge 2: Data Shareability

Scientific workflows deploy advanced algorithmic techniques in their tasks to transform the data and produce scientific discovery. These techniques leverage parallelism to improve performance. For example, due to the rapid growth of AI and ML and the significant amounts of data required to train a model and apply inference to new datasets, scientific workflows require parallel techniques for distributing data and models.

In Figure 25, we present a common workflow when training ML models leveraging data parallelism. In this case, the data is split into batches, and the same Neural Network (NN) model runs the forward and backward pass on a single batch, computing the gradients. Once completed, all the NN models send the gradients to the next stage, where it aggregates and averages to update the weights. When deploying node-local storage, the training data can be fully loaded and cached in a single node or loaded per batch on each node where it trains. If it is the first case, the NN models executed on different nodes run do not find the data and run into an error. Additionally, if all models write the gradients to node-local SSD, the update weights stage only has access to the gradients and data present in the node where this task is being executed. Consequently, the training stage runs into several errors, stopping the execution of the workflow. Data and model parallelism techniques are present in different kinds of workflows, for example, MPI-based, MapReduce, and simulation-based. Enabling shareability across pods hosted in different nodes when orchestrating

data on top of node-local storage is essential for the execution of workflows that deploy parallelism techniques and require high I/O performance.

5.5 Persistent and Shared Data with Node-Local Storage in Hybrid Cloud

We design a novel software architecture that takes into consideration the application and systems level. At the application level, we automate the orchestration of scientific workflows in hybrid cloud with no instrumentation to the original code. At the system level, we enable the orchestrated workflow to leverage node-local storage in hybrid cloud while addressing the persistence and shareability challenges. In this section, we present the design of our architecture, the interaction between components, and our implementation in hybrid cloud.

5.5.1 Orchestrating Scientific Workflows

We enable iterative development and orchestration of end-to-end scientific workflows in hybrid cloud. We leverage the customization of hybrid cloud to add functionalities through operators. These operators are application-specific controllers that extend the functionality of hybrid cloud clusters (Kubernetes and OpenShift) to create, configure, and manage services and their components running in the cluster. Specifically, we integrate pipeline operators to provide continuous integration and continuous deployment (CI/CD) to automate application building, testing, and deployment of scientific workflows in hybrid cloud.

Part of our effort is to require no instrumentation to the scientific workflow. We do not change the existing workflow code but rather build on top of it by (i) modeling the workflow as a DAG and (ii) constructing the tasks (nodes) and data (edges) as part of the pipeline operator. We load the data and tasks in the pipeline operator's domain-specific language (DSL) and specify the order of how the tasks are called in

the pipeline flow. We add the data connections (input and output data), compute and memory requirements, the container image with all the required software stack, and the type of storage (tier 0 or 1 or a combination of both) for each task. We use the pipeline operator's SDK (software development kit) to dynamically translate the pipeline from the domain-specific language into hybrid cloud cluster resources (e.g., pods, PVCs, PVs). For example, each task is translated as a pod. Finally, we obtain a hybrid cloud-defined workflow that the pipeline operator deploys in the cluster depending on the available resources.

5.5.2 Persisting Data

We draw inspiration from burst buffers (i.e., DataWarps, Data Rabbits) and design an operator (PerSSD: Persistent, Shared, and Scalable Data) that plays as the burst buffer layer that asynchronously tracks and transfers the data from node-local SSD to persistent storage (i.e., object storage). Our operator is compatible with and cooperates with the pipeline operator. PerSSD watches and tracks all the pods initiated by the pipeline operator. When the pods are completed and satisfy the controlling logic, they are transferred from node-local storage to persistent storage.

We present the general logic of PerSSD in the Algorithm 1. PerSSD initiates and runs while a pipeline workflow is executed in the hybrid cloud cluster. During initialization, our operator launches an empty database that is filled out during the workflow execution to keep track of the pods and identify when to transfer the data. The database includes metadata information from the pod and workflow execution's DAG, including:

- *ID*: Automatic unique identifier assigned to the pod
- *pod name*: Unique name of the pod assigned by the pipeline operator to pod_{ID}
- *task name*: Workflow task executed in the pod
- *node*: Node where the pod is executed

- *children*: List of children pods (that depend on the output data) of pod_{ID} in the workflow DAG
- *outputs*: Output data from pod_{ID}
- *pushed*: Boolean that indicates if the outputs of pod_{ID} have been transferred to persistent storage
- *time*: Timestamp to when the pod finishes and starts transferring the data.

After initialization, PerSSD runs in a control loop, receiving updates of the pods' statuses (*Initializing, Pending, Running, Completed*. When a pod completes with no errors, PerSSD extracts metadata information (podname, taskname, node, children, outputs), adds an entry to the database with this information, and verifies if the data can be transferred. The verification process consists of going through all the entries in the database (all the pods completed with no failure) and selecting those whose pushed flag is set to False. It iterates over that selected list of pods whose data is still in node-local storage; for each pod, it corroborates if its pod's children are in the database. If all pods are in the database, that means that those pods have also been completed, triggering a green flag for transferring the data to persistent storage. If the pod does not have children, it triggers the green flag for transferring data.

When PerSSD identifies the pods whose outputs are ready for transfer to persistent storage, it launches a job in the node where the pod was executed and moves the data from the node-local to the object storage. After the data transfer job is complete, the pushed flag in the database is updated to $True$. This logic repeats until the pipeline workflow is completed. We note that as PerSSD tracks and transfers the data from node-local to persistent storage, it automatically frees space in the node-local SSD, controlling the storage device space available for the workflow.

Algorithm 1 PerSSD Controller's Logic

1: **if** $pipelineworkflow$ starts **then**
2: Initialize database
$db \leftarrow [ID, podname, taskname, node, children, outputs, pushes, time]$
3: **while** $pipelineworkflow$ runs **do**
4: $status \leftarrow watch(pods)$
5: **if** $status(pod_{ID}) =' Completed'$ **then**
6: $db \leftarrow metadata(pod_{ID})$
7: Verify data transfer
8: **for** pod in db with $pushed = False$ **do**
9: **if** all $children(pod)$ in db or
10: no $children(pod)$ **then**
11: Create transfer job
12: $db[pod_{ID}][pushed] \leftarrow True$

5.5.3 Sharing Data

To address the lack of shareability of node-local storage across nodes in the cluster, we set up a file system with sharing protocols to connect the pods depending on the data requirements. We explore the integration of a traditional HPC parallel file system (e.g., BeeGFS [109], GekkoFS [110], etc.) on top of the NVMe across nodes; however, the integration into the cloud requires significant effort. Even though there are existing hybrid cloud operators for these distributed file systems that aim to facilitate the installation process, they are in the initial development stages. We set up a network file system (NFS) on top of NVMe and connect all pods required to share data across the network. We select NFS because is simple, allows the user to select existing storage devices, and allows multiple nodes to be mounted simultaneously.

5.5.4 Our Software Architecture

The architecture of PerSSD consists of three components: the workflow pods, the Kubernetes operator, and the NFS server on top of the node-local SSD. The workflow tasks are orchestrated using ODH, Kubeflow, and Tekton Pipelines. The operator

watches the workflow pods; after they are completed, it triggers transfer data pods to move the data from node-local storage to persistent storage (i.e., object storage). On top of node-local storage, the NFS serves as the shared file system across all workflow pods. Figure 26 presents the implementation of our approach on Kubernetes and Openshift clusters. A key aspect of our architecture is using a node in the cluster as the control plane, meaning that it manages the components of our architecture. This node hosts the operator and the NFS server; thus, we allocate more resources to this node compared to the other worker nodes, so the architecture performs as expected.

We use the Open Data Hub Pipeline Operator [111] to generate the workflow pods as part of the pipeline. We select Open Data Hub (ODH) because is intended for hybrid cloud to bridge the gap between application developers and scientists by blending the leading open-source AI tools with a unifying and intuitive user experience. ODH provides a Data Science Pipelines Operator (DSPO) that brings Kubeflow Pipelines [112] and Tekton pipelines [113] together to to design, manage, track, execute, and view data-driven scientific pipelines. Kubeflow Pipelines Software Development Kit (SDK) provides a set of Python packages to build the pipeline based on your existing workflow. It offers different packages to translate the pipeline Python DSL into a workflow YAML spec for hybrid cloud resources. We select Tekton as the package for translation.

We build our operator using the Kubernetes Operator Pythonic Framework (Kopf) [114]. Kopf is a framework and a library that facilitates the design of Kubernetes operators using Python. We use state-changing handlers in Kopf to watch and detect when the workflow pods change their status. As the status changes, we take the pod metadata and extract all the necessary information to build the DAG and identify the children. To keep track of the workflow information, we use a SQLite3 database. Regarding the data transfer, we use the Python Kubernetes API to create a job according to the controller's logic and identify the node-local and cloud object storage PVCs to attach to the job.

We deploy the NFS server using a Kubernetes deployment and configure it to store data in the node-local SSD. In addition to deployment, we create a Kubernetes service exposing the network to the cluster. Once the NFS server is a cluster object, we can access it through the PV and PVC. In the PV, we specify the NFS server's IP and then connect it from the PVC so the workflow pods that need to share data can use the claim to access the server.

5.6 Orchestrating Persistent, Shared, and Scalable Use Case Workflows with Node-Local Storage

We evaluate the workflow performance (I/O and overall execution time) deploying our software architecture. We present two testing scenarios: (i) using a benchmark and (ii) deploying a real scientific workflow. We measure how our PerSSD architecture scales as the workflow scales and assess the impact on the workflow's makespan when node-local data is managed effectively.

We use OpenShift as our hybrid cloud cluster for both testing scenarios with 17 VM instances. The control plane node (VM instance 1) has a profile of 32 cores and 128 GB of RAM with a node-local NVMe SSD of 600 GB. The other VM instances have 8 cores and 32 GB of RAM with NVMe SSDs of 300 GB.

5.6.1 FIO Benchmark

We demonstrate the generalizability of our approach by testing on a simulated workflow using FIO [71]. Figure 28 shows the structure of the workflow where we define a pipeline of a read task followed by a write task. We measure the I/O performance as we increase the number of parallel tasks. We choose this workflow because it provides two arenas for adaptation: the tasks and the data. The task functions like a black box, allowing it to be replaced by any real workflow task. The

size of the data can be adapted to mimic any real input and output data scenario of a workflow.

We show the tasks and data adaptation for two use cases to mimic real scientific workflows. A first use case is streaming data where the task is replaced by an I/O library such as ADIOS [115] that builds an in-memory buffer and streams the data to the server. The input and output data sizes are set to 10% of node RAM size to represent the generally largest data footprint for traditional HPC applications. A second use case is the training of an ML model where the tasks are replaced by multiple NN models that read and train on batches of data and write the weights of each trained model. The input data depends on how the scientists define the batches, and the model size depends on the number of parameters. For example, for ImageNET [116], the total data is 150 GB, and models trained on this dataset vary from 1.03M to 2440M parameters (4.12 GB to 9.8 GB in float32 representation).

We measure the I/O bandwidth of the data streaming scenario with a file size of 3.2 GB as it represents 10% of the RAM. We increase the number of pods from 4 to 64 and execute the read and write three times for each N-pod using two storage configurations: (i) cloud object storage (COS) and (ii) in our PerSSD architecture (using the NFS server) (see Figure 27). We observe that the write bandwidth, shown in Figure 27b, outperforms the read bandwidth in Figure 27a for both storage configurations. As anticipated, the NFS server utilizing top node-local storage achieves the highest write bandwidth. We also observe in Figure 27a that the PerSSD solution offers a small advantage over COS for smaller numbers of pods. As we increase the number of pods, the read bandwidth running on top of our solution becomes comparable to COS.

The PerSSD solution offers more tangible gains for the write bandwidth in Figure 27b. As we increase the number of pods, we notice more variability as we make more requests to the server, which affects the performance. Exploring the NFS server's sweet spot of performance based on parallel requests is part of the future work. With the NFS server ingress bandwidth exceeding the combined data sizes,

the node local PreSSD approach offers superior performance without sacrificing the data safety from persistence media. We quantify the time from when the workflow ends to the complete data transfer to COS. Figure 29 shows how as we increase the number of pods, meaning more data going from the node-local SSD to COS, the data transfer time slightly increases. This can be explained by the fact that even though we submit all N tasks in parallel, the Kubernetes scheduler allocates them in serial, so full synchronization of start time is not guaranteed. This means that as the pods complete execution, our PerSSD operator transfers the data asynchronously; at the end of the workflow, we are not transferring all $N \times 3.2GB$ of data.

5.6.2 SOMOSPIE Workflow

We apply our approach to SOMOSPIE [1], our workflow that generates, predicts, and analyzes high-resolution topographic data, such as terrain parameters (e.g., slope, aspect, hillshade) and soil moisture. SOMOSPIE is data-driven and has four modules: data generation, data transformation, ML modeling, and analysis. We focus on the data generation module [22] where the workflow leverages data partitioning to accelerate the computation of high-resolution terrain parameters (aspect, slope, and hillshade) from digital elevation models (DEMs), as shown in Figure 30. This workflow falls into the data transformation category 3, which goes from large input to large output data with large intermediary data. The workflow becomes more complex as the data scales. The data can scale in two ways: (i) scale out as we cover a larger region and (ii) scale up as we increase the resolution at which we generate the data. As the data scales, the workflow crops the data into more tiles to compute the terrain parameters (TP) for each tile, a computationally expensive task. Once the tiles are computed, the workflow merges and reprojects the high-resolution TPs.

We measure the overall makespan of the workflow in Figure 30 as we scale out from a state to the eastern USA. We generate TPs for four regions at a 30 m resolution and present the total size, including input, intermediate, and output data.

We execute the workflow with two storage configurations: (i) reading from and writing to COS and (ii) reading from and writing to node-local storage with our PerSSD architecture. In Figure 32 we visualize the total time that each configuration takes to execute the workflow three times. The total time is the time the workflow takes from loading the DEMs to the generation of the TPs and allocation in persistent storage. This means that the measurements for PerSSD include the time the operator takes to track the data allocated in node-local storage and move it to COS.

We observe that the total time is faster when we execute the workflow using PerSSD than when deploying COS (only tier 1). Specifically, we reduce the time by 17% for TN (17.1 GB), 26% for the three states of TN, NC, and SC (49.3 GB), 31% for all southeast states (132.5 GB), and 35% for eastern USA (425.5 GB). We note that as the data grows and the workflow becomes complex, the reduction percentage of total time by our architecture increases compared to COS, demonstrating the scalability benefits of PerSSD.

We quantify the overhead time the operator takes to transfer all the data to COS after the workflow finishes. Figure 33 presents the data transfer time for the four data scenarios. As expected, the time increases as the data increases, however, even for the largest data scenario (EAST) the data transfer time is negligible (0.33%) compared to the total time. The observations in Figures 32 and 33 confirm that our architecture enables node-local storage management that improves the I/O performance and the overall makespan of a scientific workflow while ensuring data persistence and shareability.

5.7 Discussion: Scheduling in HPC and Cloud Infrastructure

We discuss how schedulers are essential in the orchestration process of scientific workflows in HPC and cloud and how our PerSSD architecture can benefit from advanced schedulers with compute and storage management.

5.7.1 Scheduling in HPC and Cloud Infrastructure

The scheduler plays a crucial role in orchestrating scientific workflows in HPC and Cloud infrastructure because it handles how resources are shared between users in distributed infrastructures. HPC schedulers, such as Slurm [117] and LSF manage the jobs on bare-metal resources. While in the cloud, the schedulers must manage containerized jobs on virtual resources. These containerized workflows require complete container and network life-cycle orchestration for resiliency, scalability, and automation [118]. Because of containerization, cloud infrastructure requires extra layers to schedule and manage the containers. Cloud container orchestrators like Kubernetes or OpenShift provide declarative management of containers, their software-defined networks, filesystems, routes, services, and other required objects to ensure that the application state matches the user-specified state. However, the regular orchestrators and schedulers for Kubernetes and OpenShit are intended for services, therefore when scientific workflows run on the cloud the schedulers lack the HPC scheduling awareness functionalities.

Within the HPC and cloud convergence community, different efforts like Fluence [118, 119], Volcano [120], and other schedulers have been developed to enable (i) HPC batch-job scheduling (e.g., locality-aware node selection), (ii) HPC workloads or task-level scheduling, and (iii) allow HPC resource managers and Kubernetes/OpenShift to comanage a resource set. These schedulers can be plugged into the Kubernetes/Openshift container orchestrators, so instead of using the regular

Kubernetes scheduler, we leverage Fluence and Volcano's network policies and allocation algorithms to satisfy the scientific workflows' more complex characteristics. For example, if they are MPI-based workflows or in our case if they require locality-awareness.

Our software architecture, PerSSD would additionally benefit from integrating more advanced schedulers that not only manage the compute allocation but in some way ones that can facilitate the data management using node-local SSDs. We observe an example in an HPC supercomputer, El Capitan which is the National Nuclear Security Administration NNSA's first exascale machine. El Capitan uses Flux [121] as its resource manager which allocates compute and storage. Flux has an awareness of the local compute node relationship to the SSDs, which is not supported in Slurm. Integrating those functionalities into the cloud schedulers would be a huge step towards the convergence between HPC and cloud, and also for the scientific workflow community.

5.8 Compendium of Supported Functionalities

In Table 8 we highlight the functionalities of our architecture that ensure trustworthiness and scalability. We **trust** the workflow execution when we ensure that (i) the data can persist and can be accessed after the execution is completed, (ii) when we trace all the data and applications across the executions, and (iii) when we annotate the execution with provenance information. We ensure **scalability** when we share data across parallel tasks with high I/O performance leveraging node-local SSDs and when we have an elastic infrastructure that can be customized and scaled given the workflow requirements.

5.9 Summary of Achievements and Future Work

In this chapter, we present PerSSD, a novel software architecture that enables the orchestration of scientific workflows leveraging node-local storage while ensuring data persistence and shareability in hybrid cloud. Our architecture integrates (i) a pipeline operator to automate the orchestration of workflows, (ii) an operator that tracks and makes data in node-local storage persistent, and (iii) a file system that allows sharing the data in node-local storage across the workflow. We demonstrate how our architecture improves the workflow performance (I/O and overall execution time) for two testing scenarios (i) using an FIO benchmark and (ii) deploying a real scientific workflow. For the FIO scenario, we observe that our architecture provides I/O performance improvement compared to only COS, increasing the write bandwidth by up to 50%. For the earth science workflow, we reduce the workflow's makespan by up to 35% and note that as data in the workflow scales, the performance improvement by our architecture increases compared to COS, demonstrating the scalability benefits of PerSSD. For both cases, the time our architecture takes to asynchronously transfer the node-local data to COS is negligible and even when adding it up to the workflow execution time, the total time is faster than with only COS. Future work in our software architecture includes the exploration of integrating a parallel system on top of node-local storage to increase I/O performance, as well as testing our solution across different cloud vendors.

Chapter 6

Conclusions and Future Work

Scientific workflows become more complex as they leverage advanced computational techniques (big data analytics, AI, and ML) and run on distributed and heterogeneous infrastructure to automate and improve scientific processes. The codesign of workflows and infrastructures requires a start-to-finish collaboration between computational scientists and domain scientists to meet the new challenges of scientific computing. Furthermore, computing infrastructures must account for new deployment scenarios at the edge and in the cloud alongside traditional HPC data centers ensuring robust science. In this thesis, we address the needs of diverse scientific communities across different fields (earth sciences, biology, and materials sciences) to (i) identify common challenges in terms of reproducibility, scalability, and orchestration and (ii) codesign scientific workflows and converged infrastructure to develop robust science.

The first challenge we identify is the *lack of traceability, explainability, and reproducibility*. The multiple interoperable components of workflows, including the applications and data, are hard to trace. Additionally, with the rapid growth of AI, most applications integrate ML models, yet these methods have limited transparency and lack explainability. When workflow components cannot be traced and the results cannot be explained, it impedes the reproducibility of the workflow.

The second challenge is regarding the *hidden intermediate data and missing scalability in scientific workflows*. Data-driven or AI/ML workflows produce significant intermediate data that another application reuses, resulting in smaller output data products. Such multistage workflows obscure the complexity of large intermediate data; their underlying computational infrastructure can negatively affect their performance scalability.

The third challenge is the *insufficient orchestration and data management for workflows*. Scientific workflows run on distributed and heterogeneous infrastructure with data and computation requirements that make difficult their orchestration. A key aspect when orchestrating workflows is that the allocated infrastructure must ensure efficient data management. This means that the orchestration must ensure I/O performance and scalability while enabling scientists to trust the data for further scientific discovery.

6.1 Tracing, Explaining, and Reproducing Scientific Workflows

Scientists need execution environments that automatically trace data provenance and explain results through an in-depth data lineage and an execution trail to advance widespread reproducibility. To address this challenge, we provide a fine-grained containerized environment that automatically creates a record trail and data lineage of a workflow execution and seamlessly attaches it to the workflow components enabling data traceability and results explainability. We demonstrate how our environment ensures (i) traceability by integrating data and metadata, uniquely identifying workflow components, and assembling an execution record trail; (ii) explainability by linking the results to the methods, containerizing components without any code or modification, and offering programmatic metadata access; and (iii) reproducibility by automatically annotating the workflow execution, encapsulating the workflow

components used during execution, and packaging the software stack necessary to execute the workflow. We demonstrate the benefits of our environment for SOMOSPIE. Specifically, we use our environment for two use cases in which we trace back differences in predictions due to input data and ML methods. We compare our fine-grained environment with native and coarse-grained containerized environments and observe that our environment has limited overhead in terms of time (10%), storage space (5% for data containers and 30% application container), and it has significantly higher IO bandwidth, with a peak of 400 MB/s versus 50 MB/s for native.

6.2 Scaling Scientific Workflows in Converged Infrastructure

Scientists need infrastructure that efficiently writes and reads large intermediate data and automatically scales their scientific workflows' execution. We define an approach to composing scientific workflows in HPC and cloud infrastructure, focusing on I/O and data scalability. We study the composition of large-scale scientific workflows that deal with large intermediate data. We provide best practices on HPC and cloud infrastructure as services to enable the shareability and scalability of scientific workflows, increasing the productivity of scientists, and accelerating scientific discovery. We demonstrate the scalability of the infrastructures for SOMOSPIE. Specifically, we measure performance when scaling up the resolution (i.e., low to high) and scaling out the region (i.e., from a state to the entire CONUS). We reach an accumulated write bandwidth of 864.8 MB/s and 5.6 GB/s accumulated read bandwidth having 94 VM instances in LSF. We obtain 2.4 GB/s write bandwidth and 11.2 GB/s read bandwidth having 225 VM instances in K8s.

6.3 Orchestrating Scientific Workflows with Persistent and Shared Data

Scientists need automatic end-to-end orchestration of scientific workflows with efficient data management. To this end, we develop PerSSD, a novel software architecture that enables the orchestration of scientific workflows leveraging node-local storage while ensuring data persistence and shareability in hybrid cloud. Our architecture integrates (i) a pipeline operator to automate the orchestration of workflows, (ii) an operator that tracks and makes data in node-local storage persistent, and (iii) a file system that allows sharing the data in node-local storage across the workflow. We demonstrate how our architecture improves the workflow performance (I/O and overall execution time) for two testing scenarios (i) using an FIO benchmark and (ii) deploying a real scientific workflow. For the FIO scenario, we observe that our architecture provides I/O performance improvement compared to only COS (cloud object storage), increasing the write bandwidth by up to 50%. For the earth science workflow, we reduce the workflow's makespan by up to 35% and note that as data in the workflow scales, the performance improvement by our architecture increases compared to COS, demonstrating the scalability benefits of PerSSD. For both cases, the time our architecture takes to asynchronously transfer the node-local data to COS is negligible. Even when adding it to the workflow execution time, the total time is faster than with only COS.

6.4 Future Work

Beyond the work of this dissertation, we will keep contributing to the HPC and cloud converge initiative to support scientific workflows while ensuring the robustness of scientific discovery. To this end, the overarching goal is to create a unified environment, integrating our fine-grained containerized environment and its automatic annotation of provenance metadata into the HPC and cloud scalable and

converged infrastructure enabling automatic orchestration and trustworthiness of data at the production level. In other words, scientists should have a single entry point and interface to develop their containerized workflows, explore converged infrastructure optimization, and automatically orchestrate the workflow with high performance and trustworthiness in the results. An approach for this is to use the object IDs in the cloud object storage to keep track of data and build the record trail of the workflow execution using hybrid cloud as our malleable infrastructure that can run on top of HPC data centers, cloud, and the edge. To complete the whole ecosystem, we will create a catalog of containers for the applications and COS buckets for the data, that scientists can interact with, extend, share, and use to build new scientific workflows in multiple domains.

Bibliography

[1] D. Rorabaugh, M. Guevara, R. Llamas, J. Kitson, R. Vargas, and M. Taufer, "SOMOSPIE: A Modular SOil MOisture SPatial Inference Engine Based on Data-Driven Decisions," in *Proc. of the 15th International Conference on eScience (eScience)*, pp. 1–10, IEEE, 2019. 1, 2, 4, 6, 10, 11, 15, 26, 65, 73

[2] R. M. Llamas, M. Guevara, D. Rorabaugh, M. Taufer, and R. Vargas, "Spatial Gap-Filling of ESA CCI Satellite-Derived Soil Moisture Based on Geostatistical Techniques and Multiple Regression," *Remote. Sens.*, vol. 12, no. 4, p. 665, 2020. 1, 2, 10

[3] P. Olaya, S. Caíno-Lores, V. Lama, R. Patel, A. K. Rorabaugh, O. Miyashita, F. Tama, and M. Taufer, "Identifying Structural Properties of Proteins from X-ray Free Electron Laser Diffraction Patterns," in *2022 IEEE 18th International Conference on e-Science (e-Science)*, pp. 21–31, 2022. 1, 2, 65

[4] G. Channing, R. Patel, P. Olaya, A. Rorabaugh, O. Miyashita, S. Caino-Lores, C. Schuman, F. Tama, and M. Taufer, "Composable Workflow for Accelerating Neural Architecture Search Using In Situ Analytics for Protein Classification," in *Proceedings of the 52nd International Conference on Parallel Processing*, ICPP '23, p. 1, Association for Computing Machinery, 2023. 1, 2, 65

[5] N. Zhou, G. Scorzelli, J. Luettgau, R. R. Kancharla, J. J. Kane, R. Wheeler, B. P. Croom, P. Newell, V. Pascucci, and M. Taufer, "Orchestration of Materials Science Workflows for Heterogeneous Resources at Large Scale," *The*

International Journal of High Performance Computing Applications, vol. 37, no. 3-4, pp. 260–271, 2023. 1, 2

[6] R. Patel, B. Roachell, S. Caino-Lores, C. Ketron, J. Leonard, N. Tan, K. Vahi, D. Brown, E. Deelman, and T. Taufer, "Reproducibility of the First Image of a Black Hole in the Galaxy M87 from the Event Horizon Telescope (EHT) Collaboration," *IEEE Computing in Science and Engineering (CiSE)*, vol. 5, no. 24, pp. 42–52, 2022. 1, 2

[7] J. Cowls, A. Tsamados, M. Taddeo, and L. Floridi, "The ai gambit: leveraging artificial intelligence to combat climate change—opportunities, challenges, and recommendations," *AI & SOCIETY*, vol. 38, pp. 283–307, Feb 2023. 1

[8] J. Ang, A. A. Chien, S. D. Hammond, A. Hoisie, I. Karlin, S. Pakin, J. Shalf, and J. S. Vetter, "Reimagining Codesign for Advanced Scientific Computing: Report for the ASCR Workshop on Reimagining Codesign," 4 2022. 1

[9] R. F. da Silva and R. M. B. et. al., "Workflows Community Summit 2022: A Roadmap Revolution," tech. rep., Mar. 2023. 1, 3, 15

[10] M. Taufer, E. Deelman, R. F. d. Silva, T. Estrada, M. Hall, and M. Livny, "A Roadmap to Robust Science for High-throughput Applications: The Developers' Perspective," in *Proc. of the 2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 807–808, 2021. 2

[11] R. Ferreira da Silva, H. Casanova, A.-C. Orgerie, R. Tanaka, E. Deelman, and F. Suter, "Characterizing, Modeling, and Accurately Simulating Power and Energy Consumption of I/O-intensive Scientific Workflows," *Journal of Computational Science*, vol. 44, p. 101157, 2020. 4, 36

[12] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus:

a Workflow Management System for Science Automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015. 4, 17, 33, 58

[13] P. Olaya, J. Luettgau, C. Roa, R. Llamas, R. Vargas, S. Wen, I.-H. Chung, S. Seelam, Y. Park, J. Lofstead, and M. Taufer, "Enabling scalability in the cloud for scientific workflows: An earth science use case," in *Proc. of the 2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, pp. 383–393, 2023. 10, 58

[14] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter, "The Future of Scientific Workflows," *The International Journal of High Performance Computing Applications*, vol. 32, no. 1, pp. 159–175, 2018. 10

[15] R. Badia, E. Ayguade, and J. Labarta, "Workflows for Science: A Challenge When Facing the Convergence of HPC and Big Data," *Supercomput. Front. Innov.: Int. J.*, vol. 4, no. 1, p. 27–47, 2017. 10

[16] R. Ferreira da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A Characterization of Workflow Management Systems for Extreme-scale Applications," *Future Generation Computer Systems*, vol. 75, pp. 228–238, 2017. 10

[17] M. Guevara, M. Taufer, and R. Vargas, "Gap-free global annual soil moisture: 15 km grids for 1991–2018," *Earth System Science Data*, vol. 13, no. 4, pp. 1711–1735, 2021. 10

[18] USGS, "3DEP: 3D Elevation Program." https://apps.nationalmap.gov/downloader/. [Online; accessed 04-21-2023]. 11

[19] R. M. Llamas, L. Valera, P. Olaya, M. Taufer, and R. Vargas, "Downscaling Satellite Soil Moisture Using a Modular Spatial Inference Framework," *Remote Sensing*, vol. 14, no. 13, 2022. 11, 41

[20] "Soil Moisture CCI." Available at https://www.esa-soilmoisture-cci.org [Online; accessed 02-25-2023]. 11, 39

[21] NASA and Jet Propulsion Laboratory and PODAAC, "Soil Moisture Active Passive (SMAP)." https://podaac.jpl.nasa.gov/SMAP. [Online; accessed 04-10-2021]. 11

[22] C. Roa, P. Olaya, R. Llamas, R. Vargas, and M. Taufer, "GEOtiled: A Scalable Workflow for Generating Large Datasets of High-Resolution Terrain Parameters," in *Proc. of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, p. 311–312, 2023. 11, 56, 73

[23] C. Sar and P. Cao, "Lineage file system." https://crypto.stanford.edu/c̃ao/lineage, 2005. [Online; accessed 04-10-2021]. 16

[24] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-Aware Storage Systems," in *Proc. of the Annual Conference on USENIX '06 – Annual Technical Conference*, pp. 1–4, 2006. 16

[25] D. Thain and M. Livny, "Parrot: An application environment for data-intensive computing," *Scalable Computing: Practice and Experience*, pp. 9–18, 2005. 16

[26] P. J. Guo and D. Engler, "CDE: Using System Call Interposition to Automatically Create Portable Software Packages," in *Proc. of the 2011 USENIX Annual Technical Conference*, pp. 1–2, 2011. 16

[27] F. Chirigati, R. Rampin, D. Shasha, and J. Freire, "ReproZip: Computational Reproducibility With Ease," in *Proc. of the 2016 International Conference on Management of Data*, p. 2085–2088, 2016. 16

[28] H. Meng and D. Thain, "Facilitating the Reproducibility of Scientific Workflows with Execution Environment Specifications," *Procedia Computer Science*, vol. 108, pp. 705–714, 2017. 16

[29] L. Oliveira, D. Wilkinson, D. Mossé, and B. R. Childers, "Occam: Software environment for creating reproducible research," in *Proc. of the 2018 IEEE 14th International Conference on e-Science*, pp. 394–395, 2018. 16

[30] D. Wilkinson, L. Oliveira, D. Mossé, and B. Childers, "Software Provenance: Track the Reality Not the Virtual Machine," in *Proc. of the First International Workshop on Practical Reproducible Evaluation of Computer Systems*, pp. 1–6, 2018. 16

[31] P. Inc., "Pachyderm: The Data Foundation for Machine Learning." https://github.com/pachyderm/pachyderm, 2021. [Online; accessed 04-10-2021]. 16, 33

[32] T. Simko, L. Heinrich, H. Hirvonsalo, D. Kousidis, and D. Rodríguez, "REANA: A System for Reusable Research Data Analyses," *EPJ Web Conf.*, vol. 214, p. 06034, 2019. 16, 33

[33] D. Ghoshal, L. Bianchi, A. Essiari, M. Beach, D. Paine, and L. Ramakrishnan, "Science Capsule - Capturing the Data Life Cycle," *J. of Open Source Software*, vol. 6, no. 62, p. 2484, 2021. 16

[34] J. Lofstead, J. Baker, and A. Younge, "Data Pallets: Containerizing Storage for Reproducibility and Traceability," in *High Performance Computing*, pp. 36–45, 2019. 16, 19

[35] P. Ivie and D. Thain, "PRUNE: A preserving run environment for reproducible scientific computing," in *Proc. of 2016 IEEE 12th International Conference on e-Science*, pp. 61–70, 2016. 16

[36] "The Kepler Project." https://kepler-project.org, 2015. [Online; accessed 04-10-2021]. 17

[37] G. Malewicz, I. Foster, A. L. Rosenberg, and M. Wilde, "A tool for prioritizing DAGMan jobs and its evaluation," *J. of Grid Computing*, vol. 5, no. 2, pp. 197–212, 2007. 17

[38] K. M. D. Sweeney and D. Thain, "Efficient Integration of Containers into Scientific Workflows," in *Proc. of the 9th Workshop on Scientific Cloud Computing*, pp. 1–6, 2018. 18

[39] T. Renner, L. Thamsen, and O. Kao, "CoLoc: Distributed data and container colocation for data-intensive applications," in *Proc. of the 2016 IEEE International Conference on Big Data*, pp. 3008–3015, 2016. 18

[40] R. Scolati, I. Fronza, N. El Ioini, A. Samir, and C. Pahl, "A Containerized Big Data Streaming Architecture for Edge Cloud Computing on Clustered Single-board Devices," in *Closer*, pp. 68–80, 2019. 18

[41] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLOS ONE*, vol. 12, 05 2017. 22, 23

[42] T. Kluyver, B. Ragan-Kelley, *et al.*, "Jupyter Notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87 – 90, 2016. 23, 25

[43] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux J.*, vol. 2014, no. 239, 2014. 23

[44] R. Priedhorsky and T. Randles, "Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC," in *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–10, 2017. 23

[45] H. Gantikow, S. Walter, and C. Reich, "Rootless Containers with Podman for HPC," in *High Performance Computing*, pp. 343–354, 2020. 23

[46] C. Clerget and I. Kaneshiro, "Singularity Image Bind Support." https://github.com/apptainer/singularity/pull/4948. [Online; accessed 08-20-2022]. 23

[47] P. Olaya and D. Kennedy, "Containerized Environment." https://github.com/TauferLab/ContainerizedEnv. [Online; accessed 08-20-2022]. 24

[48] D. Kennedy, P. Olaya, J. Lofstead, R. Vargas, and M. Taufer, "Augmenting Singularity to Generate Fine-grained Workflows, Record Trails, and Data Provenance," in *Proc. of the 18th International Conference on e-Science (e-Science)*, pp. 403–404, IEEE, 2022. 24, 38, 60

[49] R. P. Paul Walsh, "NetworkX Viewer." https://github.com/jsexauer/networkx_viewer. [Online; accessed 11-10-2021]. 26

[50] D. Y. Hancock, J. Fischer, J. M. Lowe, W. Snapp-Childs, M. Pierce, S. Marru, J. E. Coulter, M. Vaughn, B. Beck, N. Merchant, E. Skidmore, and G. Jacobs, "Jetstream2: Accelerating cloud computing via Jetstream.," in *Proc. of the Practice and Experience in Advanced Research Computing (PEARC '21)*, pp. 1–8, 2021. 30

[51] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: current status and future plans," in *Proc. of the Linux symposium*, pp. 21–33, 2007. 32

[52] "Kubernetes." https://github.com/kubernetes/kubernetes. [Online; accessed 09-22-2023]. 33, 38, 58

[53] Q. Jiang, Y. C. Lee, M. Arenaz, L. M. Leslie, and A. Y. Zomaya, "Optimizing Scientific Workflows in the Cloud: A Montage Example," in *2014 IEEE/ACM*

7th International Conference on Utility and Cloud Computing, pp. 517–522, IEEE/ACM, 2014. 38, 52

[54] M. J. Rosa, C. G. Ralha, M. Holanda, and A. P. Araujo, "Computational Resource and Cost prediction Service for Scientific Workflows in Federated Clouds," *Future Generation Computer Systems*, vol. 125, pp. 844–858, 2021. 38

[55] T. Reiter, P. T. Brooks†, L. Irber†, S. E. K. Joslin†, C. M. Reid†, C. Scott†, C. T. Brown, and N. T. Pierce-Ward, "Streamlining data-intensive biology with workflow systems," *GigaScience*, vol. 10, no. 1, 2021. 38

[56] M. Krämer, H. M. Würz, and C. Altenhofen, "Executing cyclic scientific workflows in the cloud," *Journal of Cloud Computing*, vol. 10, p. 25, Apr 2021. 38

[57] R. P. Abernathey, T. Augspurger, A. Banihirwe, C. C. Blackmon-Luca, T. J. Crone, C. L. Gentemann, J. J. Hamman, N. Henderson, C. Lepore, T. A. McCaie, N. H. Robinson, and R. P. Signell, "Cloud-Native Repositories for Big Scientific Data," *Computing in Science and Engineering*, vol. 23, no. 2, pp. 26–35, 2021. 38

[58] "Dask: Parallel computing with task scheduling ." https://github.com/dask/dask. [Online; accessed 09-22-2023]. 38

[59] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li, "Big Data Processing in Cloud Computing Environments," in *Proc. of the 12th International Symposium on Pervasive Systems, Algorithms, and Networks*, pp. 17–23, IEEE, 2012. 38

[60] M. Barika, S. Garg, A. Y. Zomaya, L. Wang, A. V. Moorsel, and R. Ranjan, "Orchestrating Big Data Analysis Workflows in the Cloud: Research Challenges, Survey, and Future Directions," *ACM Comput. Surv.*, vol. 52, no. 5, 2019. 38

[61] H. Bhatia, F. Di Natale, J. Y. Moon, X. Zhang, J. R. Chavez, F. Aydin, C. Stanley, T. Oppelstrup, C. Neale, S. K. Schumacher, D. H. Ahn, S. Herbein, T. S. Carpenter, S. Gnanakaran, P.-T. Bremer, J. N. Glosli, F. C. Lightstone, and H. I. Ingólfsson, "Generalizable Coordination of Large Multiscale Workflows: Challenges and Learnings at Scale," in *Proc. of the International Conference for High-Performance Computing, Networking, Storage and Analysis (SC)*, p. 1–16, ACM/IEEE, 2021. 38

[62] P. Olaya, D. Kennedy, R. Llamas, L. Valera, R. Vargas, J. Lofstead, and M. Taufer, "Building trust in earth science findings through data traceability and results explainability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 704–717, 2023. 38, 60

[63] A. Dusia, Y. Yang, and M. Taufer, "Network Quality of Service in Docker Containers," in *Proc. of the International Conference on Cluster Computing (CLUSTER)*, pp. 527–528, IEEE, 2015. 38

[64] S. McDaniel, S. Herbein, and M. Taufer, "A Two-Tiered Approach to I/O Quality of Service in Docker Containers," in *Proc. of the International Conference on Cluster Computing (CLUSTER)*, pp. 490–491, IEEE, 2015. 38

[65] J. Monsalve, A. Landwehr, and M. Taufer, "Dynamic CPU Resource Allocation in Containerized Cloud Environments," in *Proc. of the International Conference on Cluster Computing (CLUSTER)*, pp. 535–536, IEEE, 2015. 38

[66] S. Herbein, A. Dusia, Ayushvand Landwehr, J. McDaniel, Seanvand Monsalve, S. R. Yang, Yangvand Seelam, and M. Taufer, "Resource Management for Running HPC Applications in Container Clouds," in *Proc. of the International Supercomputing Conference (ISC)*, pp. 261–278, Springer, 2016. 38

[67] B. Balis, A. Broński, and M. Szarek, "Auto-scaling of Scientific Workflows in Kubernetes," in *Proc. of the International Conference on Computational Science (ICCS)*, pp. 33–40, Springer, 2022. 38

[68] Z. Yang, P. Nguyen, H. Jin, and K. Nahrstedt, "MIRAS: Model-based Reinforcement Learning for Microservice Resource Allocation over Scientific Workflows," in *Proc. of the 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 122–132, IEEE, 2019. 38

[69] P. Olaya, J. Luettgau, N. Zhou, J. Lofstead, G. Scorzelli, V. Pascucci, and M. Taufer, "NSDF-FUSE: A Testbed for Studying Object Storage via FUSE File Systems," in *Proc. of the 31st International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, p. 277–278, ACM, 2022. 43, 54

[70] s3fs-fuse, "s3fs." Available at https://github.com/s3fs-fuse/s3fs-fuse, version 1.91 [Online; accessed 02-25-2023]. 43

[71] J. Axboe, "fio: Flexible I/O ." Available at https://github.com/axboe/fio, version 3.33 [Online; accessed 02-25-2023]. 46, 71

[72] Caltech-IPAC, "Montage: An astronomical image engine." Available at https://github.com/Caltech-IPAC/Montage [Online; accessed 10-05-2023]. 52

[73] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, *A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis*, pp. 39–59. Springer London, 2007. 52

[74] USC Epigenome Center and the Pegasus Team, "Epigenomics Workflow." Available at https://pegasus.isi.edu/workflow_gallery/gallery/epigenomics/index.php [Online; accessed 10-05-2023]. 52

[75] K. Bousselmi, S. Ben Hamida, and M. Rukoz, "Bi-Objective CSO for Big Data Scientific Workflows Scheduling in the Cloud: Case of LIGO Workflow," in *Proc. of the 15th International Conference on Software Technologies*, pp. 615–624, SCITEPRESS - Science and Technology Publications, 2020. 52

[76] N, Sadhasivam and R, Balamurugan and M, Pandi, "Cancer Diagnosis Epigenomics Scientific Workflow Scheduling in the Cloud Computing Environment Using an Improved PSO Algorithm," *Asian Pacific Journal of Cancer Prevention*, vol. 19, no. 1, pp. 243–246, 2018. 52

[77] V. Bucur, C. Dehelean, and L. Miclea, "Object Storage in the Cloud and Multi-cloud: State of the Art and the Research Challenges," in *Proc. of the 2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, pp. 1–6, 2018. 56, 59

[78] E. Larsonneur, J. Mercier, N. Wiart, E. L. Floch, O. Delhomme, and V. Meyer, "Evaluating Workflow Management Systems: A Bioinformatics Use Case," in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 2773–2775, 2018. 58

[79] R. Filgueira, R. F. Da Silva, A. Krause, E. Deelman, and M. Atkinson, "Asterism: Pegasus and Dispel4py Hybrid Workflows for Data-Intensive Science," in *2016 Seventh International Workshop on Data-Intensive Computing in the Clouds (DataCloud)*, pp. 1–8, 2016. 58

[80] L. Wratten, A. Wilm, and J. Göke, "Reproducible, Scalable, and Shareable Analysis Pipelines with Bioinformatics Workflow Managers," *Nature Methods*, vol. 18, pp. 1161–1168, Oct 2021. 58

[81] W. Gerlach, W. Tang, A. Wilke, D. Olson, and F. Meyer, "Container Orchestration for Scientific Workflows," in *2015 IEEE International Conference on Cloud Engineering*, pp. 377–378, 2015. 58

[82] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud Container Technologies: A State-of-the-Art Review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, 2019. 58

[83] P. Moreno, L. Pireddu, P. Roger, N. Goonasekera, E. Afgan, M. van den Beek, S. He, A. Larsson, D. Schober, C. Ruttkies, D. Johnson, P. Rocca-Serra, R. J. Weber, B. Gruening, R. M. Salek, N. Kale, Y. Perez-Riverol, I. Papatheodorou, O. Spjuth, and S. Neumann, "Galaxy-kubernetes integration: scaling bioinformatics workflows in the cloud," *bioRxiv*, 2019. 58

[84] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow Enables Reproducible Computational Workflows," *Nature Biotechnology*, vol. 35, pp. 316–319, Apr 2017. 58

[85] K. Vahi, M. Rynge, G. Papadimitriou, D. A. Brown, R. Mayani, R. Ferreira da Silva, E. Deelman, A. Mandal, E. Lyons, and M. Zink, "Custom Execution Environments with Containers in Pegasus-Enabled Scientific Workflows," in *2019 15th International Conference on eScience (eScience)*, pp. 281–290, 2019. 58

[86] P. Kacsuk, G. Kecskemeti, A. Kertesz, Z. Nemeth, A. Visegradi, and M. Gergely, "Infrastructure Aware Scientific Workflows and Their Support by a Science Gateway," in *2015 7th International Workshop on Science Gateways*, pp. 22–27, 2015. 58

[87] S. Singhal, A. Sussman, M. Wolf, K. Mehta, and J. Y. Choi, "DYFLOW: A Flexible Framework for Orchestrating Scientific Workflows on Supercomputers," in *50th International Conference on Parallel Processing Workshop*, ICPP Workshops '21, Association for Computing Machinery, 2021. 58

[88] I. Taylor, M. Shields, I. Wang, and A. Harrison, *The Triana Workflow Environment: Architecture and Applications*, pp. 320–339. Springer London, 2007. 58

[89] Y. Zhao, Y. Li, I. Raicu, S. Lu, W. Tian, and H. Liu, "Enabling scalable scientific workflow management in the Cloud," *Future Generation Computer Systems*, vol. 46, pp. 3–16, 2015. 58

[90] Y. Hu, H. Wang, and W. Ma, "Intelligent cloud workflow management and scheduling method for big data applications," *Journal of Cloud Computing*, vol. 9, p. 39, Jul 2020. 58

[91] Z. Ahmad, B. Nazir, and A. Umer, "A fault-tolerant workflow management system with quality-of-service-aware scheduling for scientific workflows in cloud computing," *International Journal of Communication Systems*, vol. 34, no. 1, p. e4649, 2021. 58

[92] The Galaxy Community, "The Galaxy Platform for Accessible, Reproducible and Collaborative Biomedical Analyses: 2022 Update," *Nucleic Acids Research*, vol. 50, pp. W345–W351, 04 2022. 58

[93] "Apache Airflow." https://github.com/apache/airflow. [Online; accessed 09-25-2023]. 58

[94] K. Bateman, N. Rajesh, J. Cernuda Garcia, L. Logan, J. Ye, S. Herbein, A. Kougkas, and X.-H. Sun, "LuxIO: Intelligent Resource Provisioning and Auto-Configuration for Storage Services," in *Proc. of the 2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pp. 246–255, 2022. 59

[95] D. Ghoshal and L. Ramakrishnan, "MaDaTS: Managing Data on Tiered Storage for Scientific Workflows," in *Proc. of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, p. 41–52, 2017. 59

[96] F. Tessier, M. Martinasso, M. Chesi, M. Klein, and M. Gila, "Dynamic Provisioning of Storage Resources: A Case Study with Burst Buffers," in *Proc. of the 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1027–1035, 2020. 59

[97] C. Daley, D. Ghoshal, G. Lockwood, S. Dosanjh, L. Ramakrishnan, and N. Wright, "Performance characterization of scientific workflows for the optimal use of Burst Buffers," *Future Generation Computer Systems*, vol. 110, pp. 468–480, 2020. 59

[98] R. Ferreira da Silva, S. Callaghan, T. M. A. Do, G. Papadimitriou, and E. Deelman, "Measuring the impact of burst buffers on data-intensive scientific workflows," *Future Generation Computer Systems*, vol. 101, pp. 208–220, 2019. 59

[99] A.-A. Corodescu, N. Nikolov, A. Q. Khan, A. Soylu, M. Matskin, A. H. Payberah, and D. Roman, "Big Data Workflows: Locality-Aware Orchestration Using Software Containers," *Sensors*, vol. 21, no. 24, 2021. 59

[100] M. Adel Serhani, H. T. El-Kassabi, K. Shuaib, A. N. Navaz, B. Benatallah, and A. Beheshti, "Self-adapting cloud services orchestration for fulfilling intensive sensory data-driven IoT workflows," *Future Generation Computer Systems*, vol. 108, pp. 583–597, 2020. 59

[101] C. Haine, U.-U. Haus, M. Martinasso, D. Pleiter, F. Tessier, D. Sarmany, S. Smart, T. Quintino, and A. Tate, "A Middleware Supporting Data Movement in Complex and Software-Defined Storage and Memory Architectures," pp. 346–357, 2021. 59

[102] A. Pasdar, Y. C. Lee, and K. Almi'ani, "Hybrid Scheduling for Scientific Workflows on Hybrid clouds," *Computer Networks*, vol. 181, 2020. 59

[103] X. Cai, M. Li, Y. Zhang, T. Zhao, W. Zhang, and J. Chen, "Multitasking Bi-level Evolutionary Algorithm for Data-intensive Scientific Workflows on Clouds," , vol. 238, p. 121833, 2024. 59

[104] T. Yoshimura, T. Chiba, S. Choochotkaew, S. Seelam, H. fang Wen, and J. Pfefferle, "Objcache: An Elastic Filesystem over External Persistent Storage for Container Clusters," 2023. 59

[105] "Openebs." https://github.com/openebs. [Online; accessed 05-08-2024]. 59

[106] N. by Seqera, "Fusion file system." https://seqera.io/fusion/. [Online; accessed 05-08-2024]. 59

[107] N. Tan, R. Bird, G. Chen, and M. Taufer, "Optimize memory usage in vector particle-in-cell (vpic) to break the 10 trillion particle barrier in plasma simulations," in *Computational Science – ICCS 2021*, pp. 452–465, 2021. 61

[108] N. Tan, R. F. Bird, G. Chen, S. V. Luedtke, B. J. Albright, and M. Taufer, "Analysis of Vector Particle-In-Cell (VPIC) memory usage optimizations on cutting-edge computer architectures," *Journal of Computational Science*, vol. 60, p. 101566, 2022. 61

[109] "Beegfs." hwww.beegfs.io/c/. [Online; accessed 05-08-2024]. 69

[110] M.-A. Vef, N. Moti, T. Süß, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann, "GekkoFS - A Temporary Distributed File System for HPC Applications," in *Proc. of the 2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 319–324, 2018. 69

[111] "Open data hub plaform." https://github.com/opendatahub-io/data-science-pipelines-operator . [Online; accessed 05-08-2024]. 70

[112] "Kubeflow." https://github.com/kubeflow/kubeflow. [Online; accessed 09-25-2023]. 70

[113] "Tekton pipelines." https://github.com/tektoncd/pipeline. [Online; accessed 09-29-2023]. 70

[114] S. Vasilyev, "Kubernetes operator pythonic framework (kopf)." https://github.com/nolar/kopf . [Online; accessed 05-08-2024]. 70

[115] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)," in *Proc. of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, p. 15–24, 2008. 72

[116] "Imagenet: A large-scale hierarchical image database, author=Deng, Jia and Dong, Wei and Socher, Richard and Li, Li-Jia and Li, Kai and Fei-Fei, Li," in *Proc. of the 2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009. 72

[117] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Proc. of the Job Scheduling Strategies for Parallel Processing*, (Berlin, Heidelberg), pp. 44–60, 2003. 75

[118] C. Misale, M. Drocco, D. J. Milroy, C. E. A. Gutierrez, S. Herbein, D. H. Ahn, and Y. Park, "It's a Scheduling Affair: GROMACS in the Cloud with the KubeFlux Scheduler," in *Proc. of the 2021 3rd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, pp. 10–16, 2021. 75

[119] C. Misale, D. J. Milroy, C. E. A. Gutierrez, M. Drocco, S. Herbein, D. H. Ahn, Z. Kaiser, and Y. Park, "Towards Standard Kubernetes Scheduling Interfaces for Converged Computing," in *Proc. of the Driving Scientific and Engineering Discoveries Through the Integration of Experiment, Big Data, and Modeling and Simulation*, pp. 310–326, 2022. 75

[120] Volcano, "Cloud native computing foundation (cncf)." https://github.com/volcano-sh/volcano. [Online; accessed 06-10-2024]. 75

[121] D. H. Ahn, J. Garlick, M. Grondona, D. Lipari, B. Springmeyer, and M. Schulz, "Flux: A Next-Generation Resource Management Framework for Large HPC Centers," in *Proc. of the 2014 43rd International Conference on Parallel Processing Workshops*, pp. 9–17, 2014. 76

Appendices

A Figures and Tables

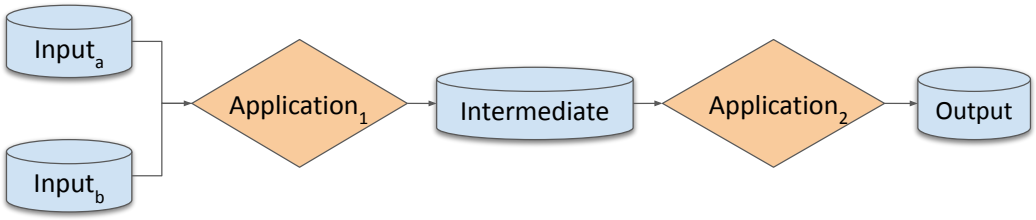


Figure 1: Structure of a general scientific workflow with one or multiple interoperable applications with input, intermediate, and output data.

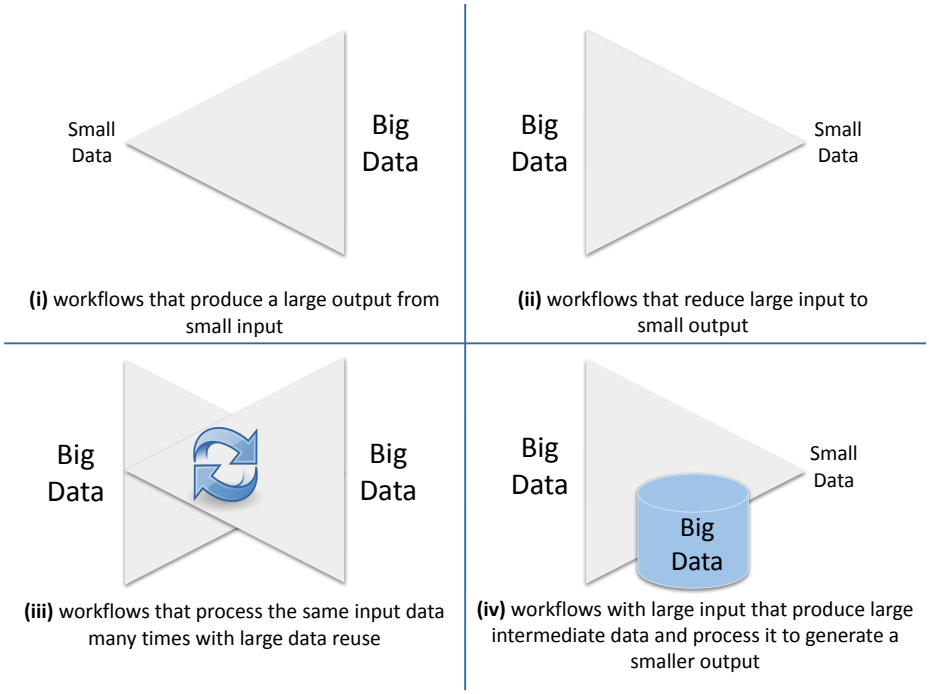


Figure 2: Modalities of data used in scientific workflows (Courtesy of Frank Wuerthwein, SDSC).

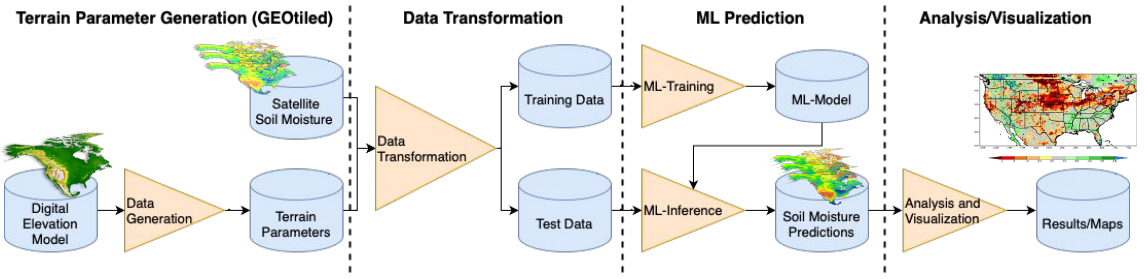
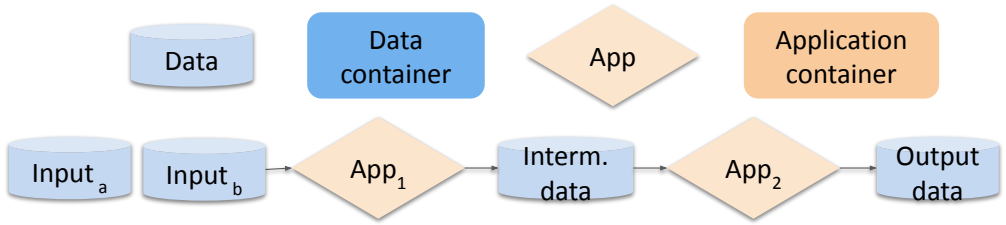
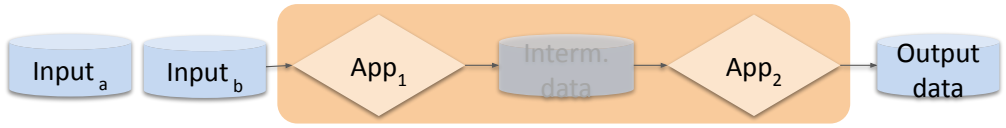


Figure 3: SOMOSPIE's modular workflow for predicting soil moisture is composed of four modules (i.e., terrain parameters generation, data transformation, ML prediction, and analysis).

(a) Native



(b) Coarse-grained



(c) Fine-grained

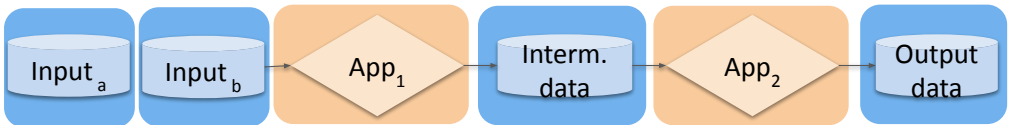
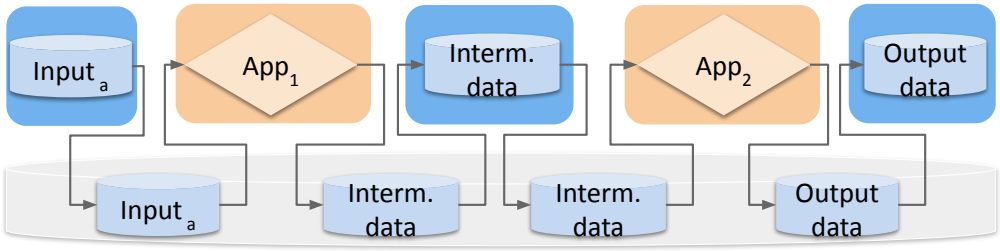


Figure 4: Composition of scientific workflows on (a) native environment (i.e., original workflow execution on backbone HPC or cloud platforms), (b) using a coarse-grained containerized environment with a single monolithic container, and (c) using our proposed fine-grained containerized environment (i.e., decoupled workflow components in data and application independent containers).

(a) Two-copy data transfer



(b) Zero-copy data transfer

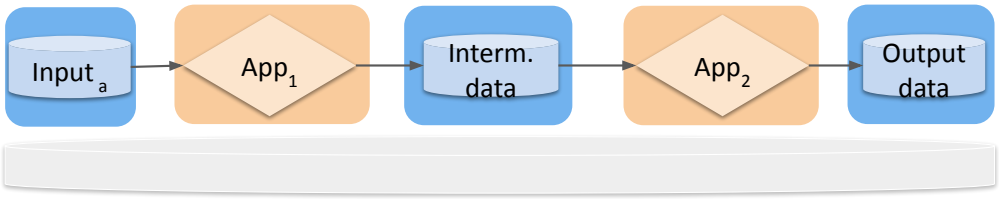


Figure 5: Communication between the workflow components in (a) two-copy (b) and zero-copy data transfer for our fine-grained containerized environment.

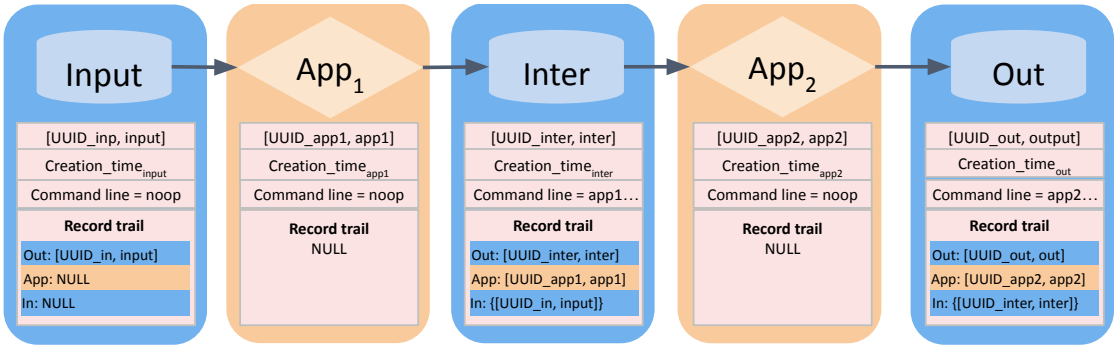
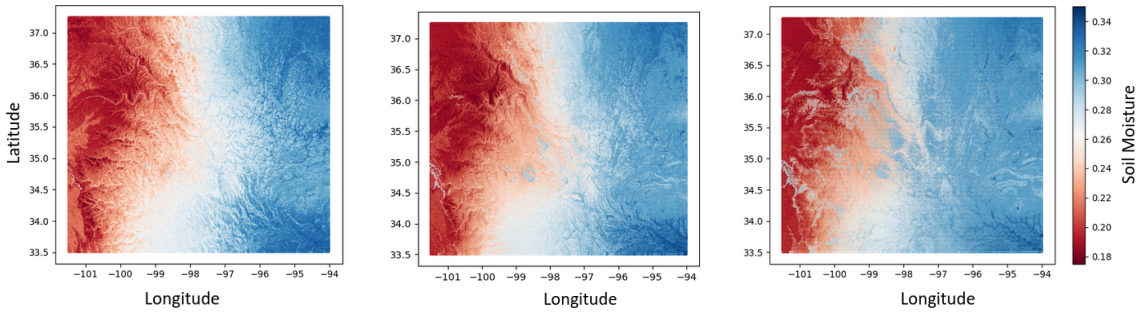
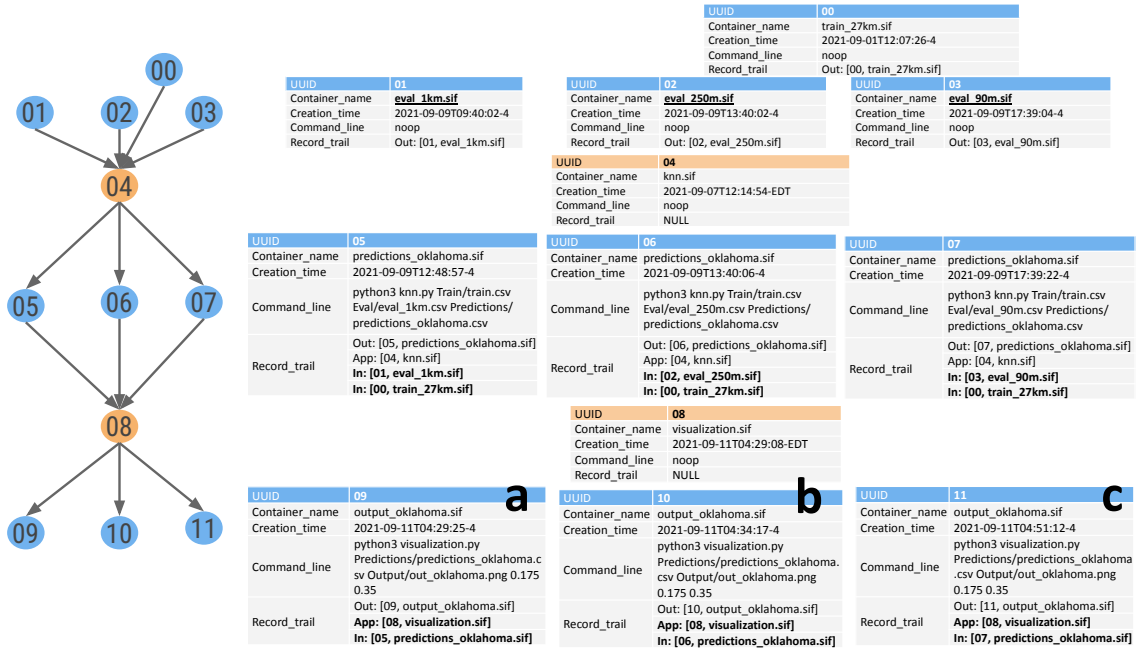


Figure 6: Example of the metadata partitions for a workflow in our fine-grained containerized environment, composed of three data containers serving as input (*Input*), intermediate (*Inter*), and output (*Out*) respectively, and two application containers (App_1 and App_2). The partitions are populated with both static and dynamic information.



(a) 1 km (b) 250 m (c) 90 m

Figure 7: Example of soil moisture output visualization for Oklahoma predicted on three different resolutions (i.e., 1 km, 250 m, and 90 m) generated with SOMOSPIE.



UUID	00
Container_name	train_27km.sif
Creation_time	2021-09-01T12:07:26-4
Command_line	noop
Record_trail	Out: [00, train_27km.sif]

UUID	01
Container_name	**eval_1km.sif**
Creation_time	2021-09-09T09:40:02-4
Command_line	noop
Record_trail	Out: [01, eval_1km.sif]

UUID	02
Container_name	**eval_250m.sif**
Creation_time	2021-09-09T13:40:02-4
Command_line	noop
Record_trail	Out: [02, eval_250m.sif]

UUID	03
Container_name	**eval_90m.sif**
Creation_time	2021-09-09T17:39:04-4
Command_line	noop
Record_trail	Out: [03, eval_90m.sif]

UUID	04
Container_name	knn.sif
Creation_time	2021-09-07T12:14:54-EDT
Command_line	noop
Record_trail	NULL

UUID	05
Container_name	predictions_oklahoma.sif
Creation_time	2021-09-09T12:48:57-4
Command_line	python3 knn.py Train/train.csv Eval/eval_1km.csv Predictions/ predictions_oklahoma.csv
Record_trail	Out: [05, predictions_oklahoma.sif] App: [04, knn.sif] **In: [01, eval_1km.sif]** **In: [00, train_27km.sif]**

UUID	06
Container_name	predictions_oklahoma.sif
Creation_time	2021-09-09T13:40:06-4
Command_line	python3 knn.py Train/train.csv Eval/eval_250m.csv Predictions/ predictions_oklahoma.csv
Record_trail	Out: [06, predictions_oklahoma.sif] App: [04, knn.sif] **In: [02, eval_250m.sif]** **In: [00, train_27km.sif]**

UUID	07
Container_name	predictions_oklahoma.sif
Creation_time	2021-09-09T17:39:22-4
Command_line	python3 knn.py Train/train.csv Eval/eval_90m.csv Predictions/ predictions_oklahoma.csv
Record_trail	Out: [07, predictions_oklahoma.sif] App: [04, knn.sif] **In: [03, eval_90m.sif]** **In: [00, train_27km.sif]**

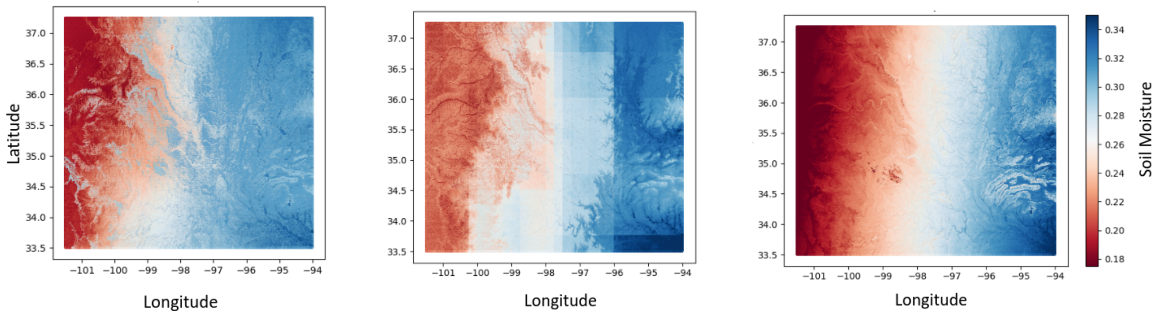
UUID	08
Container_name	visualization.sif
Creation_time	2021-09-11T04:29:08-EDT
Command_line	noop
Record_trail	NULL

UUID	09
Container_name	output_oklahoma.sif
Creation_time	2021-09-11T04:29:25-4
Command_line	python3 visualization.py Predictions/predictions_oklahoma.csv Output/out_oklahoma.png 0.175 0.35
Record_trail	Out: [09, output_oklahoma.sif] **App: [08, visualization.sif]** **In: [05, predictions_oklahoma.sif]**

UUID	10
Container_name	output_oklahoma.sif
Creation_time	2021-09-11T04:34:17-4
Command_line	python3 visualization.py Predictions/predictions_oklahoma.csv Output/out_oklahoma.png 0.175 0.35
Record_trail	Out: [10, output_oklahoma.sif] **App: [08, visualization.sif]** **In: [06, predictions_oklahoma.sif]**

UUID	11
Container_name	output_oklahoma.sif
Creation_time	2021-09-11T04:51:12-4
Command_line	python3 visualization.py Predictions/predictions_oklahoma.csv Output/out_oklahoma.png 0.175 0.35
Record_trail	Out: [11, output_oklahoma.sif] **App: [08, visualization.sif]** **In: [07, predictions_oklahoma.sif]**

Figure 8: Graph representation of the SOMOSPIE workflow executions for Oklahoma on three resolutions (i.e., 1 km, 250 m, and 90 m) generated by the Jupyter Notebook interface. Each table in the figure presents the metadata for each containerized component.

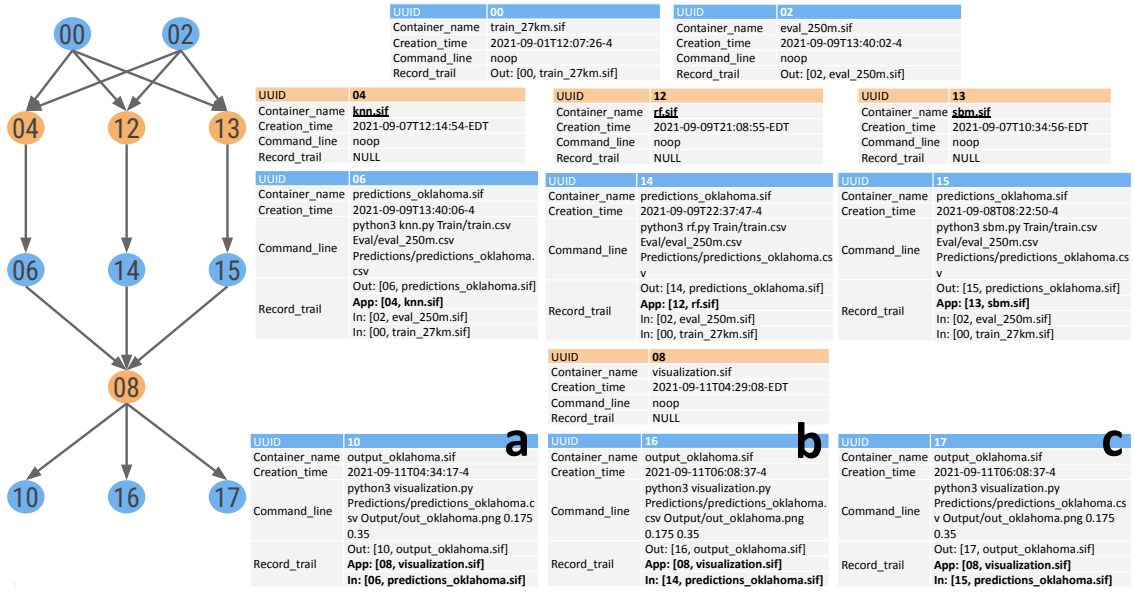


(a) K-Nearest Neighbors (KNN)

(b) Random Forest (RF)

(c) Surrogate-based Modeling (SBM)

Figure 9: Example of soil moisture output visualization for Oklahoma predicted with three different ML methods (i.e., KNN, RF, and SBM) generated with SOMOSPIE.



UUID	00
Container_name	train_27km.sif
Creation_time	2021-09-01T12:07:26-4
Command_line	noop
Record_trail	Out: [00, train_27km.sif]

UUID	02
Container_name	eval_250m.sif
Creation_time	2021-09-09T13:40:02-4
Command_line	noop
Record_trail	Out: [02, eval_250m.sif]

UUID	04
Container_name	knn.sif
Creation_time	2021-09-07T12:14:54-EDT
Command_line	noop
Record_trail	NULL

UUID	12
Container_name	rf.sif
Creation_time	2021-09-09T21:08:55-EDT
Command_line	noop
Record_trail	NULL

UUID	13
Container_name	sbm.sif
Creation_time	2021-09-07T10:34:56-EDT
Command_line	noop
Record_trail	NULL

UUID	06
Container_name	predictions_oklahoma.sif
Creation_time	2021-09-09T13:40:06-4
Command_line	python3 knn.py Train/train.csv Eval/eval_250m.csv Predictions/predictions_oklahoma.csv
Record_trail	Out: [06, predictions_oklahoma.sif] **App: [04, knn.sif]** In: [02, eval_250m.sif] In: [00, train_27km.sif]

UUID	14
Container_name	predictions_oklahoma.sif
Creation_time	2021-09-09T22:37:47-4
Command_line	python3 rf.py Train/train.csv Eval/eval_250m.csv Predictions/predictions_oklahoma.csv
Record_trail	Out: [14, predictions_oklahoma.sif] **App: [12, rf.sif]** In: [02, eval_250m.sif] In: [00, train_27km.sif]

UUID	15
Container_name	predictions_oklahoma.sif
Creation_time	2021-09-08T08:22:50-4
Command_line	python3 sbm.py Train/train.csv Eval/eval_250m.csv Predictions/predictions_oklahoma.csv
Record_trail	Out: [15, predictions_oklahoma.sif] **App: [13, sbm.sif]** In: [02, eval_250m.sif] In: [00, train_27km.sif]

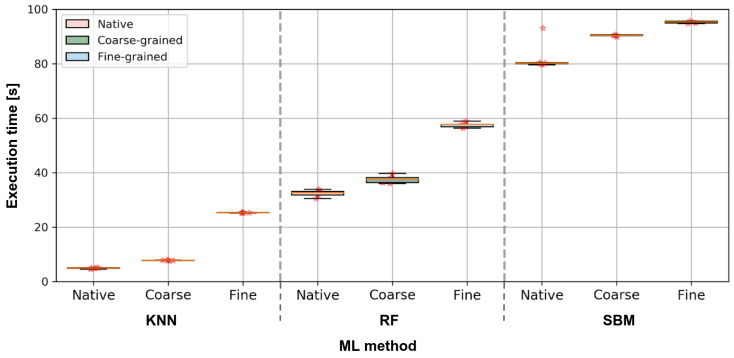
UUID	08
Container_name	visualization.sif
Creation_time	2021-09-11T04:29:08-EDT
Command_line	noop
Record_trail	NULL

UUID	10
Container_name	output_oklahoma.sif
Creation_time	2021-09-11T04:34:17-4
Command_line	python3 visualization.py Predictions/predictions_oklahoma.csv Output/out_oklahoma.png 0.175 0.35
Record_trail	Out: [10, output_oklahoma.sif] **App: [08, visualization.sif]** **In: [06, predictions_oklahoma.sif]**

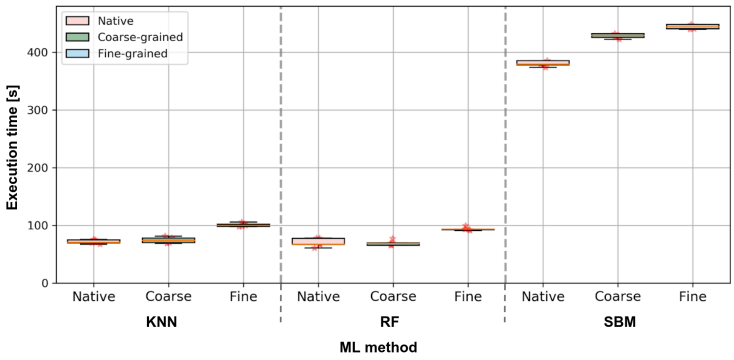
UUID	16
Container_name	output_oklahoma.sif
Creation_time	2021-09-11T06:08:37-4
Command_line	python3 visualization.py Predictions/predictions_oklahoma.csv Output/out_oklahoma.png 0.175 0.35
Record_trail	Out: [16, output_oklahoma.sif] **App: [08, visualization.sif]** **In: [14, predictions_oklahoma.sif]**

UUID	17
Container_name	output_oklahoma.sif
Creation_time	2021-09-11T06:08:37-4
Command_line	python3 visualization.py Predictions/predictions_oklahoma.csv Output/out_oklahoma.png 0.175 0.35
Record_trail	Out: [17, output_oklahoma.sif] **App: [08, visualization.sif]** **In: [15, predictions_oklahoma.sif]**

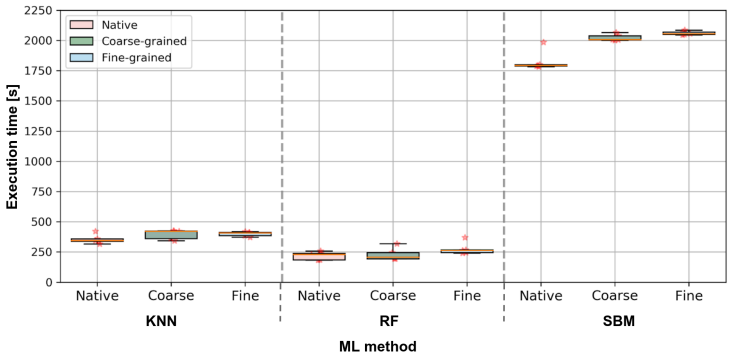
Figure 10: Graph representation of the SOMOSPIE workflow executions for Oklahoma with three ML methods (KNN, RF, and SBM) generated by the Jupyter Notebook interface. Each table in the figure presents the metadata for each containerized component.



(a) Oklahoma, 1 km

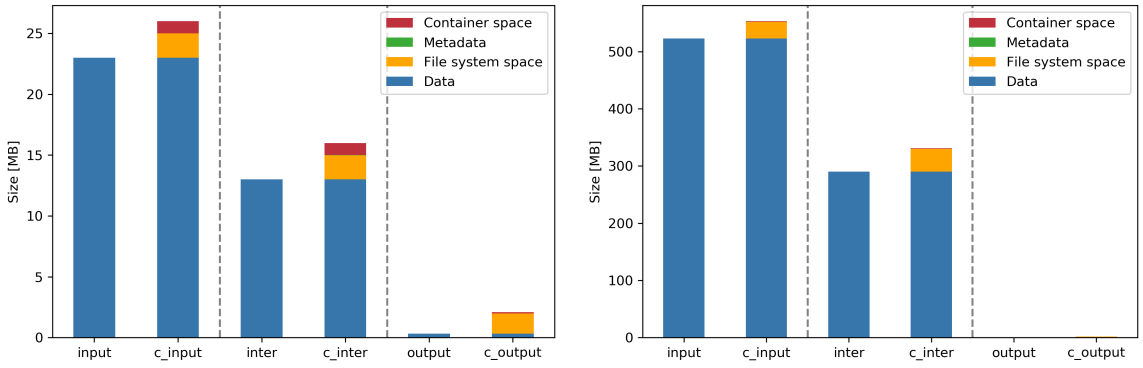


(b) Oklahoma, 250 m



(c) Oklahoma, 90 m

Figure 11: Execution time of SOMOSPIE, running on Oklahoma with three resolutions: (a) 1 km, (b) 250 m, and (c) 90 m, with three ML methods comparing the native (no containers), coarse (single container), and fine-grained containerized (multiple containers) environments.



(a) Oklahoma, 1 km (b) Oklahoma, 250 m

Figure 12: Comparison between the native and the fine-grained containerized environment for the data storage including the input, intermediate (inter), and output data of the earth science workflow running in Oklahoma with two resolutions: (a) 1 km, and (b) 250 m.

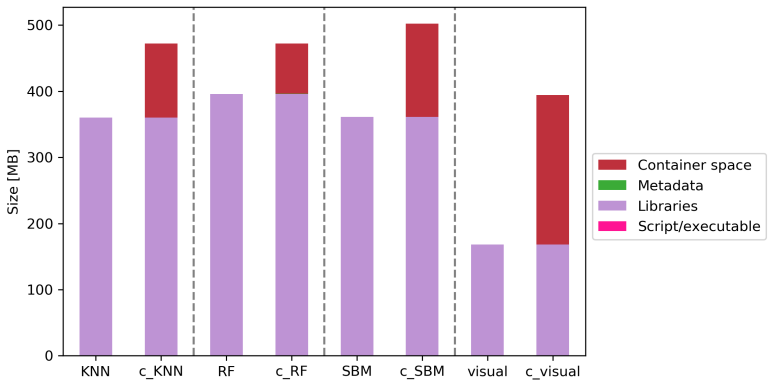
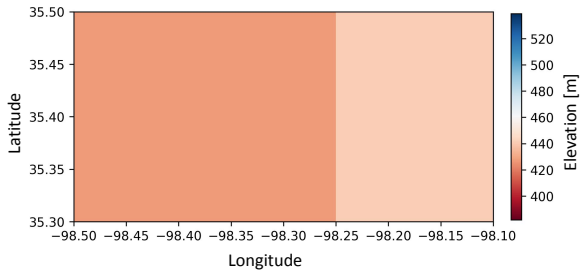


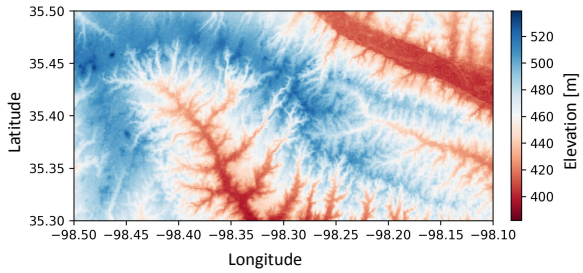
Figure 13: Comparison between the native and the fine-grained containerized environment for the application's storage in SOMOSPIE.

Table 1: Functionalities from our environment that ensure the replicability, reproducibility, and traceability of scientific workflows.

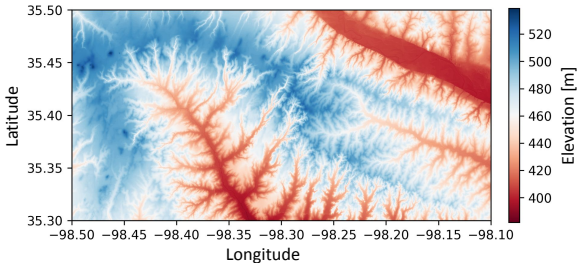
Supported functionality	**Description**
Traceability	
Integration of data and metadata	Tight relation between the data and its metadata by collocating the metadata file as a partition of the output data container
Record trail assembly	Complete identification of the interaction between components inside the workflow (data and apps)
Unique identification of components	Encapsulation of the files in containers; Universally unique identifiers for each container
Explainability and Transparency	
Results linked to methods	Concise information of the application containers used in execution to generate new data and command line with hyper-parameters
Simple tool integration	Containerized framework that requires no modifications on the application
Programmatic metadata access	Direct manipulation of each container and the execution parameters of the application through Apptainer `sif` tool
Replicability and Reproducibility	
Automatic workflow metadata	Automatic documentation of inputs, applications, outputs, and command line after each execution
Unique identification of components	Encapsulation of the files in containers; universally unique identifiers for each container
Packaging software system	Application container includes the software package which has all the system tools, libraries, and system settings



(a) Midwest at 27 km (satellite resolution), 2×1 data points grid in the selected area

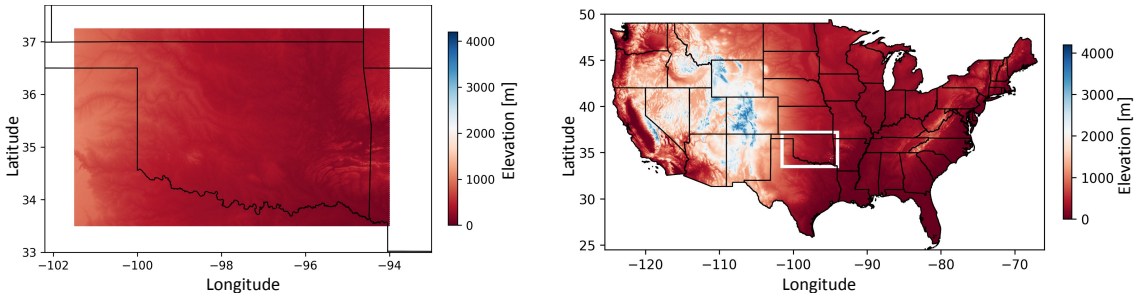


(b) Midwest at 90 m, 453×227 data points grid in the selected area



(c) Midwest at 10 m, 8926×4539 data points grid in the selected area

Figure 14: Example of satellite resolution at (a) 27 km, (b) high resolution of 90 m, and (c) a higher resolution at 10 m for a selected area in the Midwest region. (Scalability: Same region but higher resolution.)



(a) Midwest at 10 m (b) CONUS at 10 m

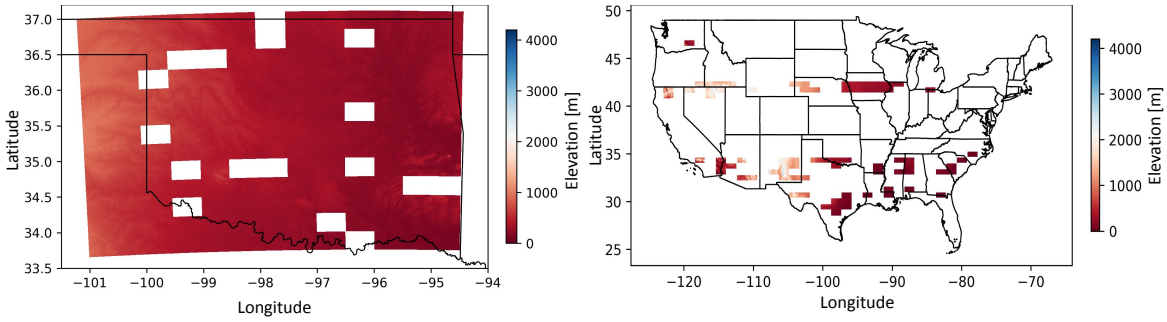
Figure 15: Example of (a) a state region, Midwest at 10 m and (b) whole country region, Contiguous United States (CONUS) at 10 m. (Scalability: Same resolution but a larger region.)

Table 2: Data scenarios for SOMOSPIE. Resolution scalability: from the Midwest region at 90 m to the Midwest region at 10 m. Region scalability: from the Midwest region at 10 m to CONUS at 10 m.

Region Resolution	Data Type	Data Description	Data Characterization	
			Points [#]	Size [B]
Midwest, 90 m x 90 m	Input	Satellite data, 27 km x 27 km	450	44,198
	Input	Terrain params (4 params), 90 m x 90 m	36,073,181	2.43 G
	Intermediate	1 prediction, Midwest, 90 m x 90 m	36,073,181	1.42 G
	Output	1 visualization, Midwest, 90 m x 90 m	-	438,126
Total Midwest 90 m x 90 m				**4.25 G**
Midwest, 10 m x 10 m	Input	Satellite data, 27 km x 27 km	450	44,198
	Input	Terrain params (4 params), 10 m x 10 m	2,921,927,661	248.51 G
	Intermediate	1 prediction, Midwest, 10 m x 10 m	2,921,927,661	135.90 G
	Output	1 visualization, Midwest, 10 m x 10 m	-	4.57 M
Total Midwest 10 m x 10 m				**388.98 G**
CONUS, 10 m x 10 m	Input	Satellite data, 27 km x 27 km	8,214	985,600
	Input	Terrain params (4 params), 10 m x 10 m	99,925,046,500	9.00 T
	Intermediate	1 prediction, CONUS, 10 m x 10 m	99,925,046,500	5.12 T
	Output	1 visualization, CONUS, 10 m x 10 m	-	21.14 M
Total CONUS 10 m x 10 m				**14,934.00 G**

Table 3: Data partitioning for SOMOSPIE scenarios.

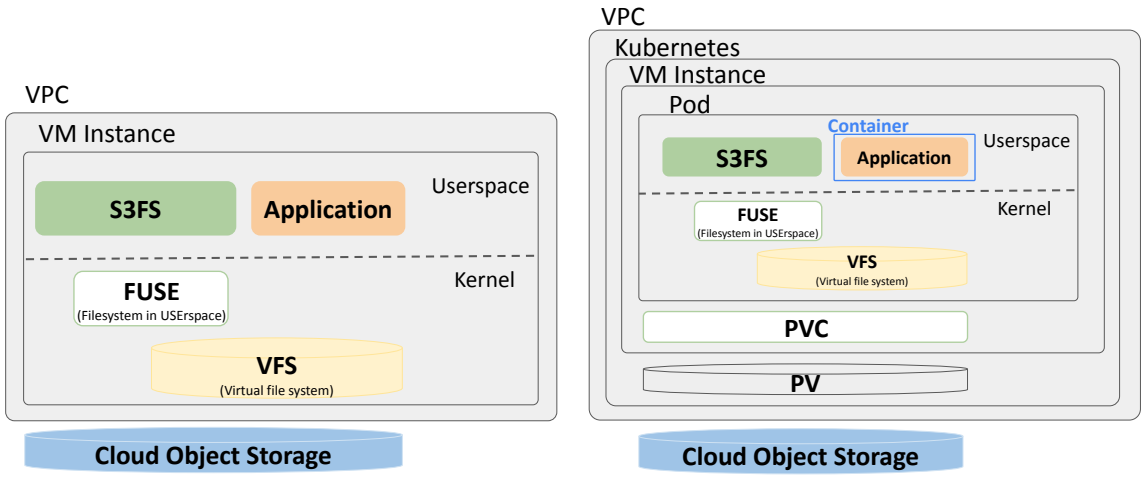
Region, Resolution	**Data Type**	**Data Description**	**Tiles$_{total}$**	**Size per tile [B]**
Midwest,	Input	Terrain params	1	2.43 G
90 m x 90 m	Intermediate	1 prediction	1	1.42 G
Midwest,	Input	Terrain params	225	900 M - 1.6 G
10 m x 10 m	Predictions	1 prediction	225	350 M - 800 MB
CONUS,	Input	Terrain params	1,156	1.1 G - 11 G
10 m x 10 m	Predictions	1 prediction	1,156	650 MB - 6.1 G



(a) 152 tiles of the same size for the Midwest at 10 m

(b) 152 tiles of the same size for CONUS at 10 m

Figure 16: Example of (a) a state region, Midwest at 10 m and (b) whole country region, Contiguous United States (CONUS) at 10 m. (Scalability: Same resolution but a larger region.)



(a) IBM Spectrum LSF (LSF)

(b) Kubernetes (K8s)

Figure 17: Architectures of the LSF (HPC) and Kubernetes (cloud-native) services with object storage to handle the data transformations for a single VM instance.

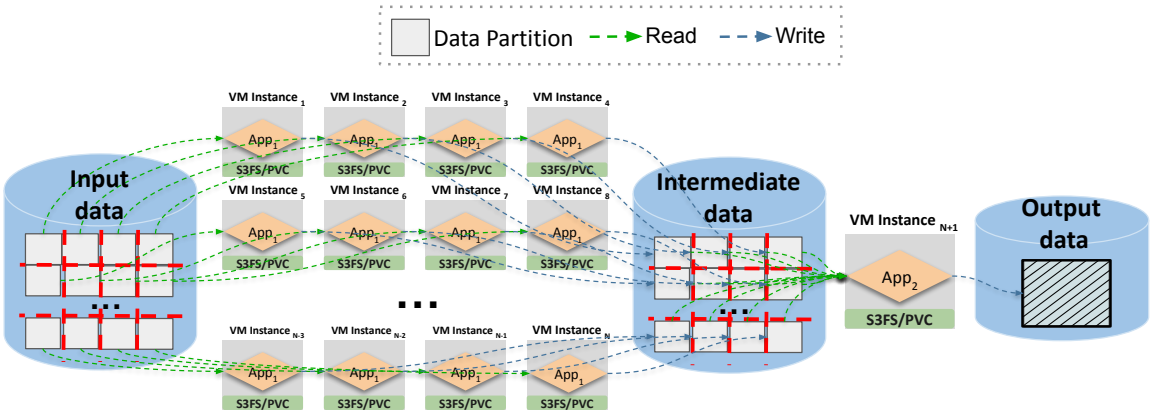
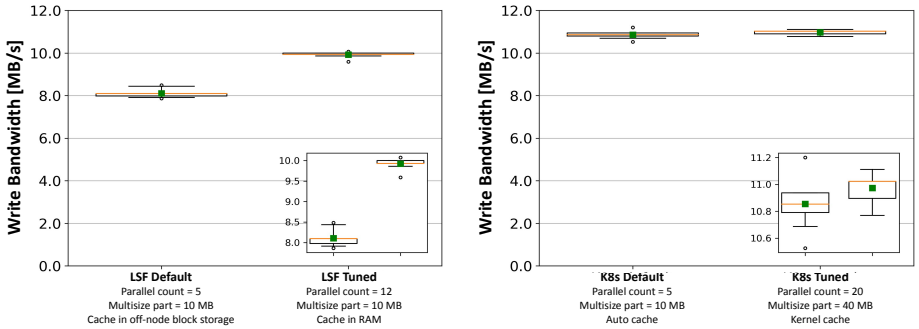
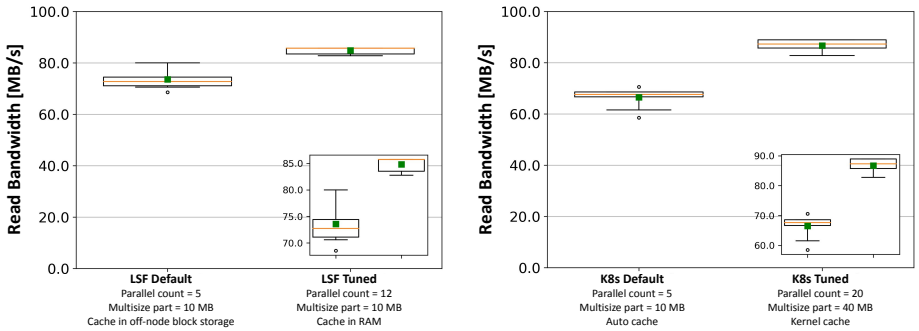


Figure 18: Composition of a workflow with large intermediate data (fourth modality) using the cloud services (i.e., VM instances from each HPC on cloud service connected to three different COS buckets through S3FS).



(a) Write bandwidth on LSF (b) Write bandwidth on Kubernetes



(c) Read bandwidth on LSF (d) Read bandwidth on Kubernetes

Figure 19: SOMOSPIE write and read bandwidth comparison on the LSF and Kubernetes deployment clusters before and after tuning the s3fs parameters when executing for the Midwest region at 90 m resolution. The smaller images zoom into the boxplots to outline the difference in I/O performance.

Table 4: Write bandwidth statistics obtained when running the FIO benchmark ten times with the default and the tuned S3FS parameters generating the best empirically observed I/O performance. PC: Parallel count. MP: Multisize part.

Cluster	**Default Parameters**	**Write Bandwidth [MB/s]**	**Tuned Parameters**	**Write Bandwidth [MB/s]**
LSF	PC = 5 MP = 10 MB Off-instance cache	median=255 mean=252 stdev=13	PC = 12 MP = 10 MB Cache in RAM	median=423 mean=406 stdev=54
Kubernetes	PC = 5 MP = 10 MB Auto cache	median=200 mean=196 stdev=14	PC = 20 MP= 40 MB Kernel cache	median=350 mean=338 stdev=39

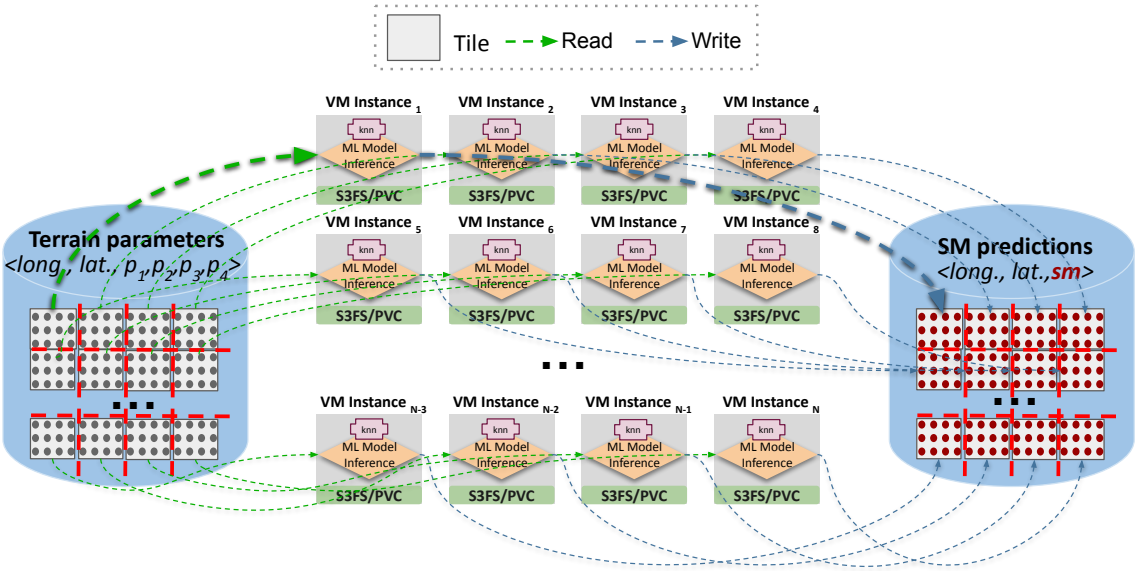
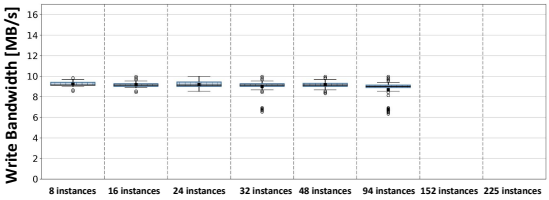
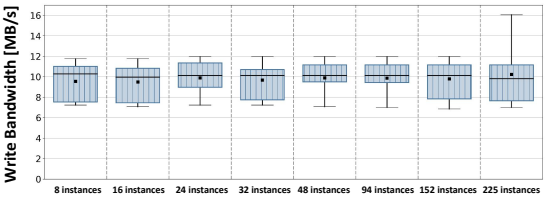


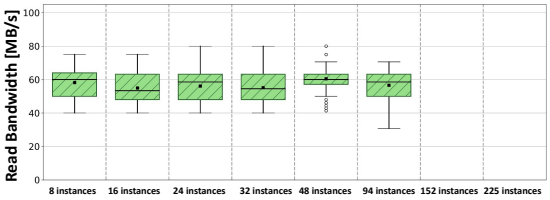
Figure 20: Multiple VM instances configuration for the Midwest region and CONUS at 10 m resolution.



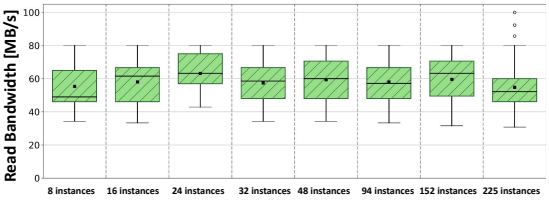
(a) Weak scale write bandwidth for Midwest at 10 m on LSF.



(b) Weak scale write bandwidth for Midwest at 10 m on Kubernetes.

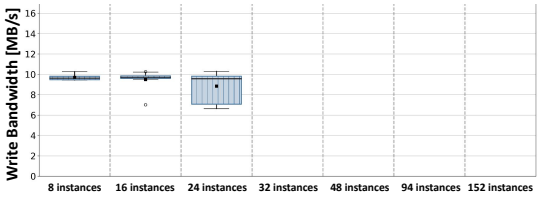


(c) Weak scale read bandwidth for Midwest at 10 m on LSF.

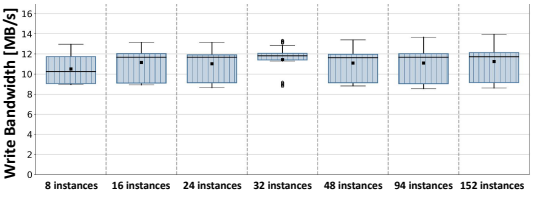


(d) Weak scale read bandwidth for Midwest at 10 m on Kubernetes.

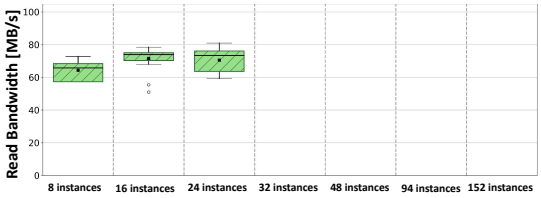
Figure 21: I/O bandwidth scalability as we increase the number of VM instances (from 8 to 225) for reading a size $1.2GB$ tile and writing $685MB$ soil moisture predictions for each tile for the Midwest region at 10 m resolution.



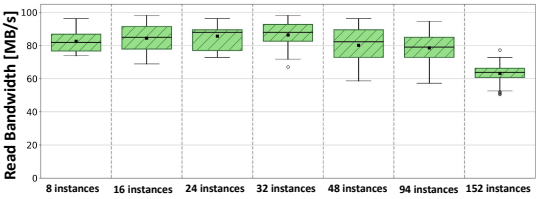
(a) Weak scale write bandwidth for CONUS at 10 m on LSF.



(b) Weak scale write bandwidth for CONUS at 10 m on Kubernetes.



(c) Weak scale read bandwidth for CONUS at 10 m on LSF.



(d) Weak scale read bandwidth for CONUS at 10 m on Kubernetes.

Figure 22: I/O bandwidth scalability as we increase the number of VM instances (from 8 to 152) for reading a tile of size 5.1 GB and writing 2.8 GB soil moisture predictions for each tile for the CONUS at 10 m resolution.

Table 5: Modeling costs apply to our three SOMOSPIE data scenarios: Midwest 90 m, Midwest 10 m, and CONUS 10m on the two deployment clusters: LSF and Kubernetes. We list the times in seconds for readability purposes, but when calculating the cost, we convert them into hours.

Cluster	Data Scenario	$tiles_{total}$	$N_{workerVMs}$	N_{cores}	RAM_{size}	$t_{tile}[s]$	$t_{total}[s]$	$cost_{total}[\$]$
LSF	Midwest 90m	1	1	4	32	331.5	331.5	0.5
	Midwest 10 m	225	94	2	16	155.7	466.8	16.0
	CONUS 10 m	1156	24	8	64	868.4	42483.0	1484.1
Kubernetes	Midwest 90m	1	1	4	32	330.9	330.6	0.2
	Midwest 10 m	225	225	2	16	153.5	153.5	12.3
	CONUS 10 m	1156	1156	8	64	984.8	984.8	1638.7

Table 6: Functionalities from our work that support first steps towards HPC and cloud convergence and workflow scalability.

Supported functionality	**Description**
HPC and Cloud Convergence	
Architecture disclosure	Identification and understanding of all architecture layers and their interactions in the HPC and cloud infrastructure
Converged technology integration	Integration of common technology and interfaces in HPC and cloud infrastructure. Allocation of data in a common technology solution
Infrastructure usability	Demonstration of technology deployment, tuning, and cost model for real scientific workflows
Scalability	
Distributed technology	Design and selection of computation and storage technology that matches the distributed requirements of the workflow
Horizontal scalability	Flexible growth of resources in both infrastructure with customizable types of VMs depending on the workflow
Infrastructure tuning	Tuning of performance parameters (i.e., multithreading, data chunking, caching) in the data architecture layers for workflows optimization

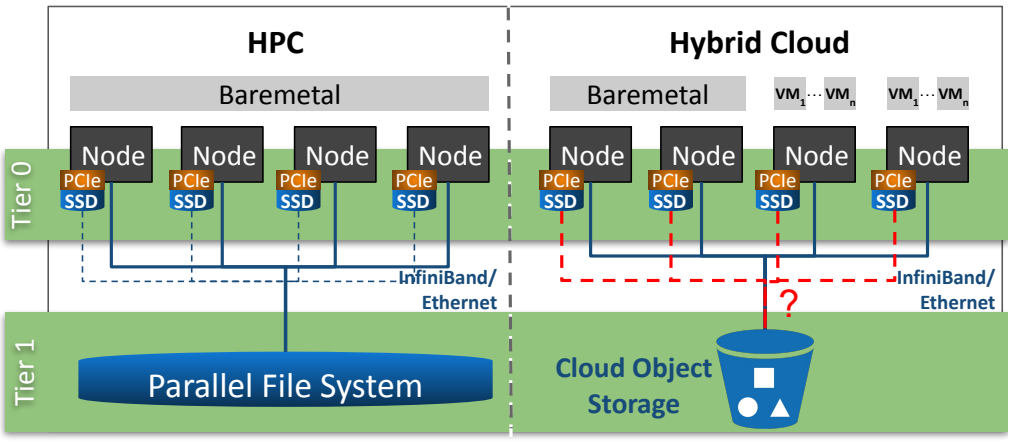


Figure 23: General infrastructure comparison between HPC and hybrid cloud highlighting the two-tiered storage architecture. For tier 1, HPC relies upon parallel file systems, while the hybrid cloud uses object storage. For tier 0, both HPC and hybrid cloud provision node-local SSD storage.

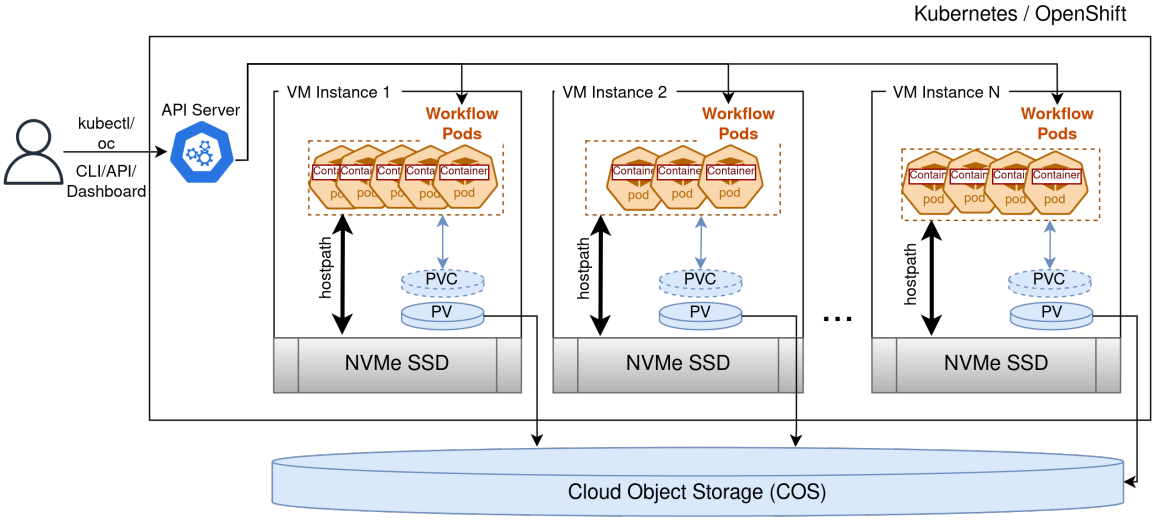


Figure 24: General Kubernetes and OpenShift cluster architecture.

Table 7: Total data (input, intermediate, and output) and number of tiles for each data scenario.

Region	Description	Total data [GB]	Number of Tiles
TN 30m	Tennessee (TN)	17.1	2
SE-3 30m	Tennessee (TN), North Carolina (NC) South Carolina (SC)	49.3	2
SE 30m	Southeast USA	132.5	3
EAST 30m	Eastern USA	425.5	4

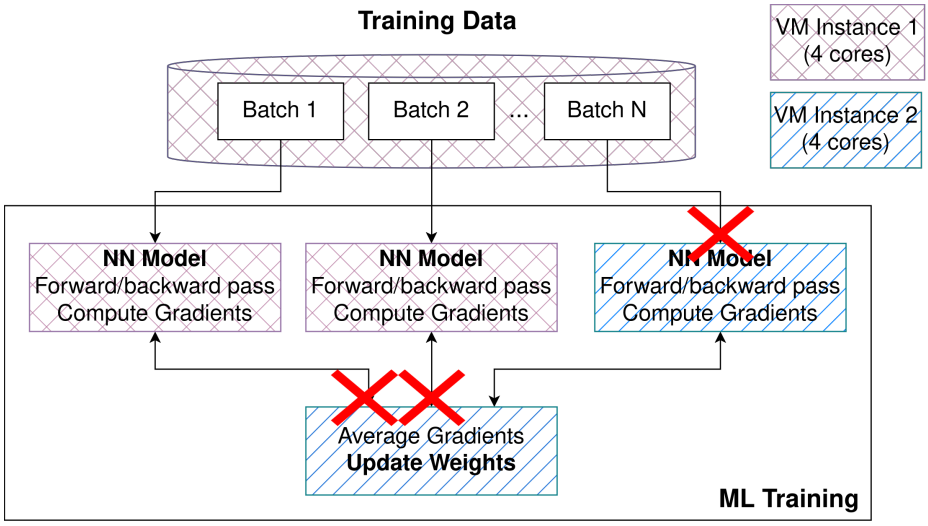


Figure 25: Demonstration of a failure when deploying a common ML training workflow pipeline using node-local storage due to the lack of shareability across nodes in the cluster.

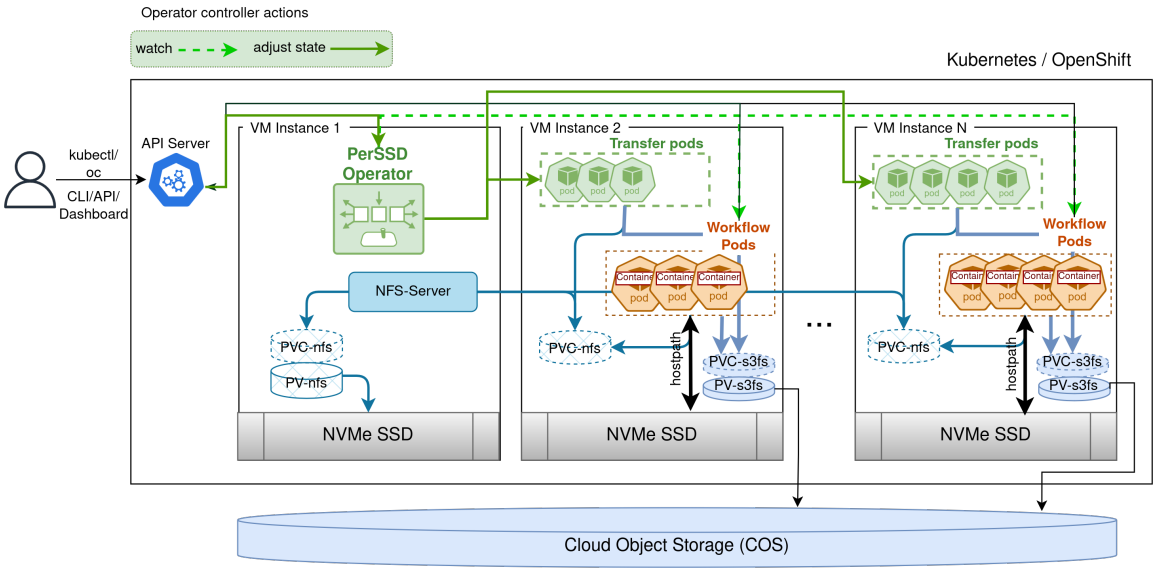
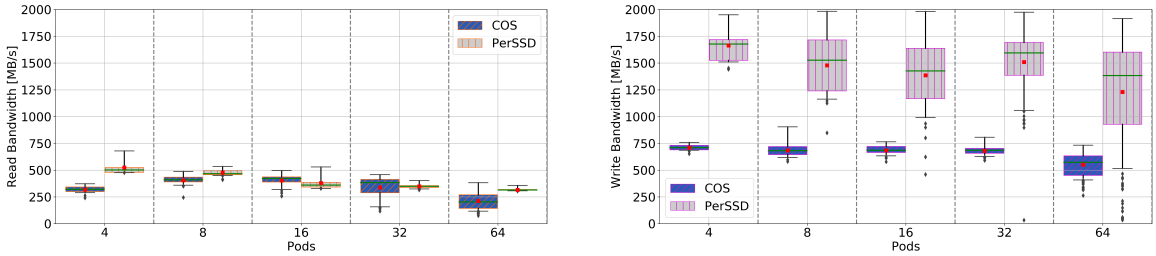


Figure 26: The architecture of PerSSD consists of three components: the workflow pods, the PerSSD operator, and the NFS server on top of the local NVMe SSD. The workflow tasks are orchestrated using ODH, Kubeflow, and Tekton Pipelines. The operator watches the workflow pods and triggers transfer data pods to move the data from node-local storage to persistent storage (i.e., object storage) after completion. The NFS server on top of node-local storage serves as the shared file system across all workflow pods.



(a) Read Bandwidth.

(b) Write Bandwidth

Figure 27: I/O bandwidth scalability comparison between COS and our PerSSD architecture (using the NFS Server) as we increase the number of pods [4,8,16,32,64], each reading a file of 3.2 G and writing a file of 3.2 GB. The 3.2 GB results from testing the 10% of the VM instances' RAM.

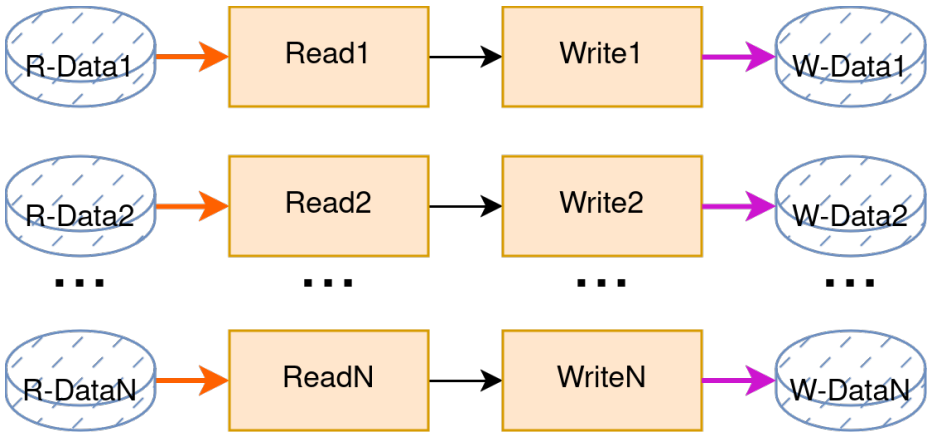


Figure 28: FIO workflow structure with N tasks in parallel that read N datasets, each with a fixed size, followed by N writing tasks to N independent files.

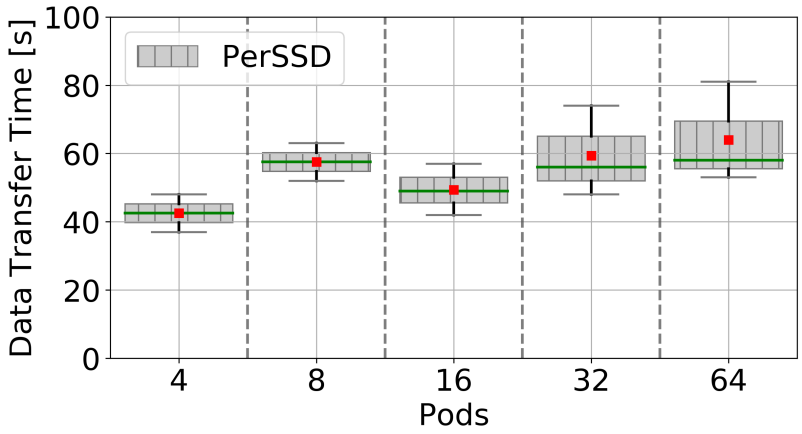


Figure 29: Transfer time of all data from node-local storage to COS after the FIO benchmark completes its execution for different pod configurations.

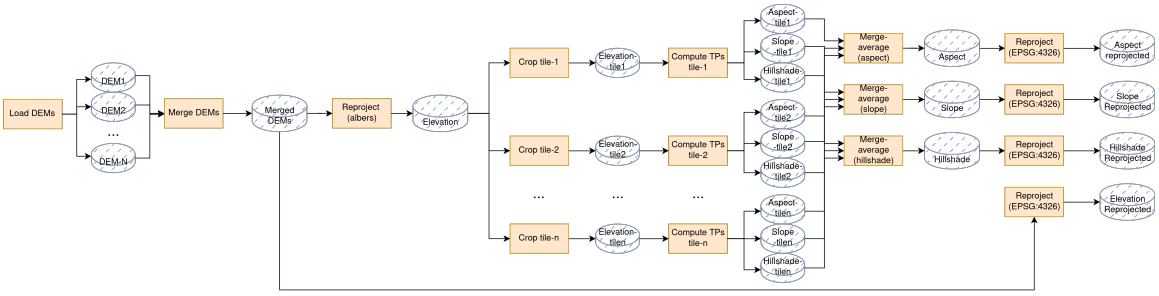


Figure 30: Earth science workflow that leverages data parallelism for the generation of high-resolution terrain parameters (aspect, slope, hillshade) from Digital Elevation Models.

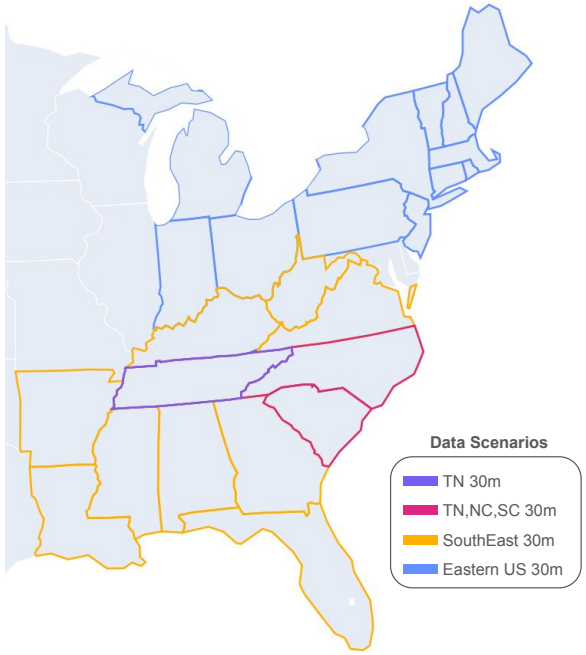


Figure 31: Data scenarios for measuring scalability as we cover a larger region (scale out). From (i) the state of TN (purple) to (ii) 3 states in the southeast: TN, NC, SC (red) to (iii) all states in the southeast (yellow) to (iv) eastern USA (blue).

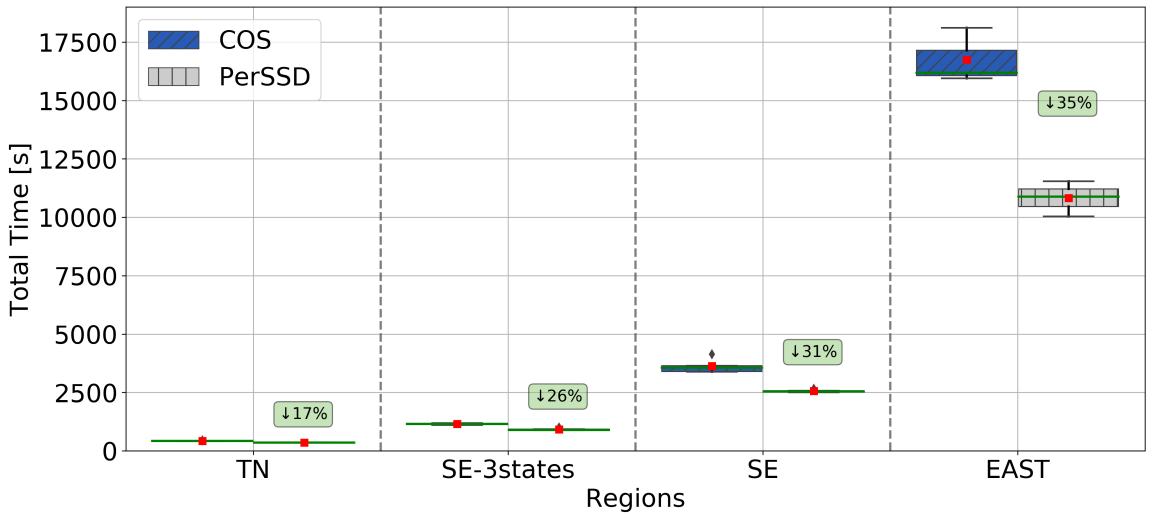


Figure 32: Total time comparison between executing the workflow deploying COS and using node-local storage with PerSSD. The total time includes the time from the start of the workflow until the allocation of all data in persistent storage.

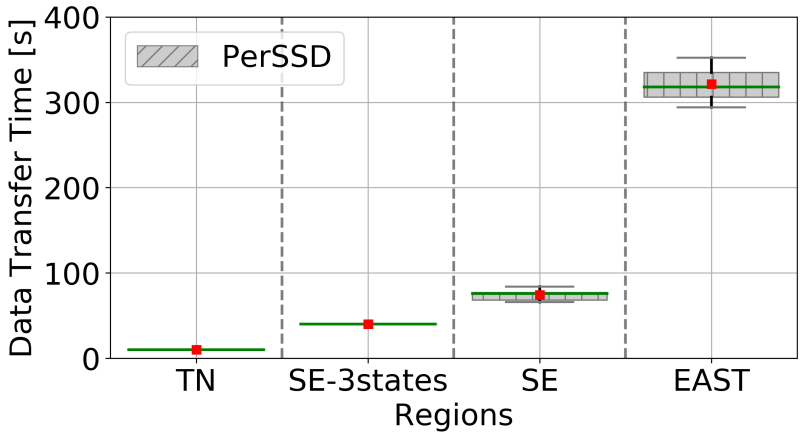


Figure 33: Data transfer time for the four data scenarios that takes the operator to transfer all the data to COS after the workflow completes.

Table 8: Functionalities from our environment that ensure trustworthy and scalable scientific workflows.

Supported functionality	**Description**
Trustworthiness	
Persisting data across the execution	Transfer of data from ephemeral to persistent storage with direct access of data during and after the workflow execution for inspection
Tracing data across the execution	Identification of the interaction between components inside the workflow (data and apps)
Collecting metadata across the execution	Creation of a database, that is updated at every job allocation cycle with DAG information about the workflow (e.g., data, task, pod, and node where the task runs, etc.)
Scalability	
Shareability of data across pods	Distributed file system that ensures the sharing of data across tasks running on different pods and nodes when using noce-local SSD
Distributed and malleable infrastructure	Execution on hybrid cloud resources that enable elasticity in the resources and automation of the tasks to scale as the workflow requires it
High I/O performance	Leverage of node-local storage technology to transfer data faster and automatically manage the available capacity given the data

B Conference and Journal Publications

P. Olaya, S. Wen, J. Lofstead, and M. Taufer, "PerSSD: Persistent, Shared, and Scalable Data with Node-Local Storage for Scientific Workflows in Hybrid Cloud". In Proceedings of the IEEE Cluster, 2024. (IN REVIEW)

J. Luettgau, H. Martinez, **P. Olaya**, G. Scorzelli, G. Tarcea, J.Lofstead, C. Kirkpatrick, V. Pascucci, and M. Taufer. "NSDF-Services: Integrating Networking, Storage, and Computing Services into a Testbed for Democratization of Data Delivery". In Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC '23). Article 11, 1–10. 2023.

C. Roa, M. Rynge, **P. Olaya**, K. Vahi, T. Miller, J. Griffioen, S. Knuth, J. Goodhue, D. Hudak, A. Romanella, R. Llamas, R. Vargas, M. Livny, E. Deelman, and M. Taufer. "End-to-end Integration of Scientific Workflows on Distributed Cyberinfrastructures: Challenges and Lessons Learned with an Earth Science Application". In Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC '23). Article 17, 1–9. 2023.

G. Channing, R. Patel, **P. Olaya**, A. Rorabaugh, O. Miyashita, S. Caino-Lores, C. Schuman, F. Tama, and M. Taufer. "Composable Workflow for Accelerating Neural Architecture Search Using In Situ Analytics for Protein Classification". In Proceedings of the 52nd International Conference on Parallel Processing (ICPP '23). 2023.

P. Olaya, J. Luettgau, C. Roa, R. Llamas, R. Vargas, S. Wen, I. Chung, S. Seelam, Y. Park, J. Lofstead, and M. Taufer, "Enabling Scalability in the Cloud for Scientific Workflows: An Earth Science Use Case". In Proceedings of the IEEE International Conference on Cloud Computing, pp. 383-393, Chicago, IL, USA, 2023.

P. Olaya, D. Kennedy, R. Llamas, L. Valera, R. Vargas, J. Lofstead, and M. Taufer, "Building Trust in Earth Science Findings through Data Traceability and Results Explainability," in IEEE Transactions on Parallel and Distributed Systems, vol. 34, no. 2, pp. 704-717, 1 Feb. 2023.

P. Olaya, S. Caino-Lores, V. Lama, R. Patel, A. Rorabaugh, F. Tama, O. Miyashita, and M. Taufer, "Identifying Structural Properties of Proteins from X-ray Free Electron Laser Diffraction Patterns". In Proceedings of the 18th IEEE International Conference on e-Science (eScience), pp. 1–10, Salt Lake City, Utah, USA, October 10-14, 2022. Best Paper Candidate.

R. M. Llamas, L. Valera, **P. Olaya**, M. Taufer, and R. Vargas, "Downscaling Satellite Soil Moisture Using a Modular Spatial Inference Framework". Remote Sensing 14, No. 13: 3137. 2022.

C Software Artifacts

Table 9: Software Artifacts

Software		
Github Repositories	**Description**	**Repository Name**
PerSSD-NVMe: Kubernetes operator for Persistent, Shared, and Scalable Data when deploying local NVMe SSDs for scientific workflows	Kubernetes/OpenShift operator that plays as the "burst buffer" layer to track and save data on persistent storage (i.e., object storage) when the application leverages the high throughput and low latency of reading and writing data from and to local NVMe SSDs.	TauferLab/perssd-nvme
Traceability and Reproducibility through Individual Containerization (TRIC)	A fine-grained containerized environment that enables data traceability and results explainability.	TauferLab/Containerized Env
SOMOSPIE (SOil MOisture SPatial Inference Engine)	ML-based earth science workflow (SOMOSPIE) that uses ML methods to predict soil moisture data from low-resolution satellite data to high-resolution values.	TauferLab/SOMOSPIE
XPSI: X-ray Free Electron Laser (XFEL) based Protein Structure Identifier	Workflow that combines deep learning and traditional machine learning to identify three structural properties (i.e., orientation, conformation, and protein type) through the diffraction patterns of a given protein.	TauferLab/XPSI

D Posters

P. Olaya and M. Taufer, "Enabling Reproducibility and Scalability of Scientific Workflows in HPC and Cloud," In Doctoral Showcase at the 2023 ACM/IEEE International Conference for High-Performance Computing, Networking, Storage and Analysis, Doctoral Showcase, Denver, CO, November 2023. ACM/IEEE Computer Society.

C. Roa, **P. Olaya**, R. Llamas, R. Vargas, and M. Taufer. "GEOtiled: A Scalable Workflow for Generating Large Datasets of High-Resolution Terrain Parameters". In Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '23). pp. 311–312. 2023. ACM.

R. Patel, A. Rorabaugh, **P. Olaya**, S. Caíno-Lores, G. Channing, C. Schuman, O. Miyashita, F. Tama, and M. Taufer. "A Methodology to Generate Efficient Neural Networks for Classification of Scientific Datasets". In Proceedings of the 18th IEEE International Conference on e-Science (eScience), pages 1–2, Salt Lake City, Utah, USA, October 10-14, 2022. IEEE Computer Society.

D. Kennedy, **P. Olaya**, J. Lofstead, and M. Taufer. "Augmenting Singularity to Generate Fine-grained Workflows, Record Trails, and Data". In Proceedings of the 18th IEEE International Conference on e-Science (eScience), pages 1–2, Salt Lake City, Utah, USA, October 10-14, 2022. IEEE Computer Society.

P. Olaya, J. Luettgau, N. Zhou, J. Lofstead, G. Scorzelli, V. Pascucci, and M. Taufer, "NSDF-FUSE: A Testbed for Studying Object Storage via FUSE File Systems". In Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing (HPDC 22), pages 1–2, Minneapolis, MN, USA. June 27-July 1, 2022. ACM.

J. Luettgau, **P. Olaya**, N. Zhou, G. Scorzelli, V. Pascucci, and M. Taufer, "NSDF-Cloud: Enabling Ad-Hoc Compute Clusters Across Academic and Commercial Clouds". In Proceedings of the 31st International Symposium on High-Performance

Parallel and Distributed Computing (HPDC 22), pages 1–2, Minneapolis, MN, USA. June 27-July 1, 2022. ACM.

P. Olaya, J. Lofstead, and M. Taufer, "Containerized Environment for Reproducibility and Traceability of Scientific Workflows," In Poster at the 2020 ACM/IEEE International Conference for High-Performance Computing, Networking, Storage and Analysis, SC20 Graduate Research Competition, Atlanta, GA, November 2020. ACM/IEEE Computer Society.

P. Olaya, M. R. Wyatt II, S. Caino-Lores, F. Tama, O. Miyashita, P. Luszczek, and M. Taufer, "XPSI: X-ray Free Electron Laser-based Protein Structure Identifier," In Poster at the 2020 ACM/IEEE International Conference for High-Performance Computing, Networking, Storage and Analysis, Research Poster, Atlanta, GA, November 2020. ACM/IEEE Computer Society.

T. Kitson, **P. Olaya**, E. Racca, M. Wyatt, M. Guevara, R.Vargas, and M. Taufer, "Data Analytics for Modeling Soil Moisture Patterns across United States Ecoclimatic Domains," In Proceedings of the 2017 IEEE International Conference on Big Data (BigData), pages 4768–4770, Boston, MA, USA, December 11-14 2017. IEEE Computer Society.

Vita

Paula Olaya is a Ph.D. student in Computer Science at the University of Tennessee, Knoxville. She earned her M.Sc. in Computer Science in Spring 2020 at UTK. Olaya's research interests in high-performance computing include the composition of scientific workflows in HPC and cloud-converged infrastructures, to allow intrinsic replicability, trustworthiness, transparency, and traceability. She has been involved in projects with HPC and machine learning applications, such as protein identification through diffraction patterns, also benchmarking and analyzing performance in terms of time and power usage for FFTs and sparse matrix solvers at the architecture and algorithm level for an astrophysics application.