PerSSD: Persistent, Shared, and Scalable Data with Node-Local Storage for Scientific Workflows in Cloud Infrastructure

Paula Olaya
University of Tennessee
Knoxville, USA
polaya@vols.utk.edu

Sophia Wen
IBM Research
Yorktown Heights, USA
hfwen@us.ibm.com

Jay Lofstead
Sandia National Laboratories
Albuquerque, USA
gflofst@sandia.gov

Michela Taufer
University of Tennessee
Knoxville, USA
mtaufer@utk.edu

2024 IEEE International Conference on Big Data (BigData) | 979-8-3503-6248-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/BigData62323.2024.10826021

Abstract—**Computational workflows need to retain data from both intermediate stages and final results to ensure the reproducibility and trustworthiness of scientific discoveries. While cloud infrastructure offers advantages like elasticity and automation, it compromises the persistence of intermediate data to ensure performance and reduce costs. Utilizing node-local storage can enhance performance but requires manual data transfers to persistent storage, making the technique impractical. To address these challenges, we propose a software architecture called Persistent, Shared, and Scalable Data (PerSSD) that integrates cloud operators and a Network File System (NFS) to make node-local data persistent and shareable across cloud nodes while ensuring performance. PerSSD outperforms traditional cloud object storage, achieving 35% reduction in the overall execution time of an earth science workflow, all while ensuring data persistence and shareability.**

I. Introduction

Scientific workflows play a crucial role in scientific discovery, enabling the transformation and analysis of large datasets across multiple stages. These workflows generate significant amounts of intermediate data vital for subsequent tasks and final outputs. However, ensuring the reproducibility and trustworthiness of these workflows is a challenge, particularly in cloud environments. While cloud infrastructure offers benefits such as elasticity, scalability, and automation, it often lacks mechanisms to guarantee data persistence and efficient data sharing across distributed tasks. Cloud object storage was originally designed for services and accessibility and not optimized for complex data transformations in scientific workflows that require high throughput and low latency when handling large data [1]. Additionally, the high throughput and low latency performance benefits of cloud-based node-local storage (e.g., SSDs) are offset by data persistence and shareability challenges across nodes.

Our paper introduces PerSSD, a novel Persistent, Shared, and Scalable Data architecture designed to address these challenges in cloud environments. The key contribution of PerSSD is its unique ability to combine the high-performance capabilities of node-local SSDs with persistent storage solutions. PerSSD ensures that data is both rapidly accessible and durably stored across the cloud infrastructure. Unlike traditional cloud storage models that treat node-local storage as ephemeral and isolated, PerSSD integrates node-local storage with persistent cloud object storage to provide a unified, scalable, and reliable data management solution.

The novelty of this paper is fourfold. First, we provide *data persistence and shareability with node-local storage in the cloud*. Cloud's node-local storage, such as NVMe SSDs, offers high performance. However, it is also ephemeral and lacks data persistence and cross-node sharing mechanisms. It presents significant challenges in multi-node workflows, where data loss or lack of access can disrupt tasks, as demonstrated in the MuMMI cancer modeling workflow [2]. PerSSD overcomes this challenge by combining Kubernetes operators with an integrated NFS, ensuring data persistence and shareability across nodes without sacrificing performance or requiring manual intervention.

Second, we deliver *cloud-native orchestration without code instrumentation*. Existing orchestration systems require code modifications or are tied to specific vendors, introducing high development overhead and limiting flexibility. Systems like Pegasus [3], [4] and Nextflow [5], [6] are not fully portable across cloud platforms. PerSSD offers a seamless cloud-native solution by integrating the Kubeflow operator, allowing workflows to be deployed across different cloud platforms without modifying the original code or facing vendor lock-in.

Third, we design a *burst buffer adaptation for cloud environments*. In HPC systems, burst buffers buffer data between computation and persistent storage, typically absent in cloud environments, leading to I/O bottlenecks when relying on object storage [7], [8]. PerSSD adapts the burst buffer concept to cloud infrastructure, asynchronously transferring data from node-local SSDs to persistent cloud storage, ensuring high performance and data persistence during workflow execution without significant latency.

Finally, we demonstrate *performance and scalability in real-world scientific workflows*. Traditional cloud object storage often becomes a bottleneck, especially in I/O-bound workflows like those in earth science. For example, workflows that generate and analyze high-resolution topographic data experience significant slowdowns due to I/O limitations [9]. PerSSD addresses these challenges, reducing execution time by 35% in an earth science workflow with 425 GB of data, demonstrating its scalability and performance for data-intensive workflows such as those in earth sciences and machine learning.

Our contributions include designing and implementing PerSSD, evaluating its performance in cloud-based scientific workflows, and exploring how our solution scales as the data and workflow complexity increase. By addressing the application needs of scientific workflows and the system challenges of the cloud, PerSSD advances cloud-based data management for scientific discovery.

II. Data Transformations on the Cloud

Data management and orchestration problems arise as scientists study new infrastructure and algorithmic techniques to optimize computational workflows. We define the data complexity of scientific workflows and stress the importance of the underlying infrastructure to enable the required performance while ensuring the workflows' trustworthiness on the cloud.

A. Scientific Workflows and Data Transformations

Data is fundamental in scientific discovery. As data scales, scientists can expand the boundaries of the discovery. Scientists design computational workflows that transform the data to obtain insights that lead to scientific breakthroughs. These computational workflows transform the data using tasks such as processing, modeling, analyzing, and visualizing tasks. The structure of a workflow can be modeled as direct acyclic graphs (DAGs) where the tasks (nodes) are interconnected by data (edges) with a specific scientific objective [10], [11]. The tasks in the workflow transform data in four different modalities [10]: 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.

As data volumes increase, workflows become more complex, yet the demands for scalability and performance persist. Alongside ensuring performance, there is a growing need within the community to maintain the trustworthiness of scientific discoveries, as noted in recent studies [12], [13]. Scientists employing these workflows to explore scientific phenomena must have trust in every component involved (i.e., data, methods, software, and hardware). To this end, we argue that it is crucial for scientists to have access to intermediate data and to trace its transformations throughout the workflow, enabling a deeper understanding of the scientific results.

B. Cloud Storage Infrastructure

The infrastructure where these 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 for addressing workflow requirements. For instance, a computation-heavy workflow can leverage running on multiple larger CPU or GPU nodes. In contrast, 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 [14], [15].

Cloud infrastructure provides on-demand, elastic, and scalable computing and storage resources for executing scientific workflows. Fig. 1 presents a diagram of cloud infrastructure, highlighting the two-tiered storage design. This storage design comprises Tier 0 with direct compute-to-storage access and Tier 1 with across-cluster data accessibility. In this paper, we expand SSD's capabilities at Tier 0 to ensure persistency and shareability, and at Tier 1, we select Cloud Object Storage (COS) to ensure scalability and distributed access to data.

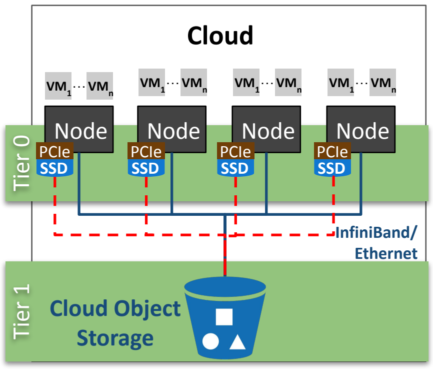


Fig. 1: Two-tiered storage architecture in cloud infrastructure where Tier 0 comprises node-local SSDs and Tier 1 consists of cloud object storage because of its distributed characteristics.

C. Cloud Orchestration

Cloud infrastructure gives scientists control over their resources and customizes them 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 cloud infrastructure is a consequence of the virtualization of the computational layers. Furthermore, container orchestration has become the standard for portable and flexible computation of scientific workflows in clouds.

Container management systems like Kubernetes and OpenShift enable users to run any workflow consistently across any footprint. In Fig. 2, 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 a cluster, and external components communicate with one

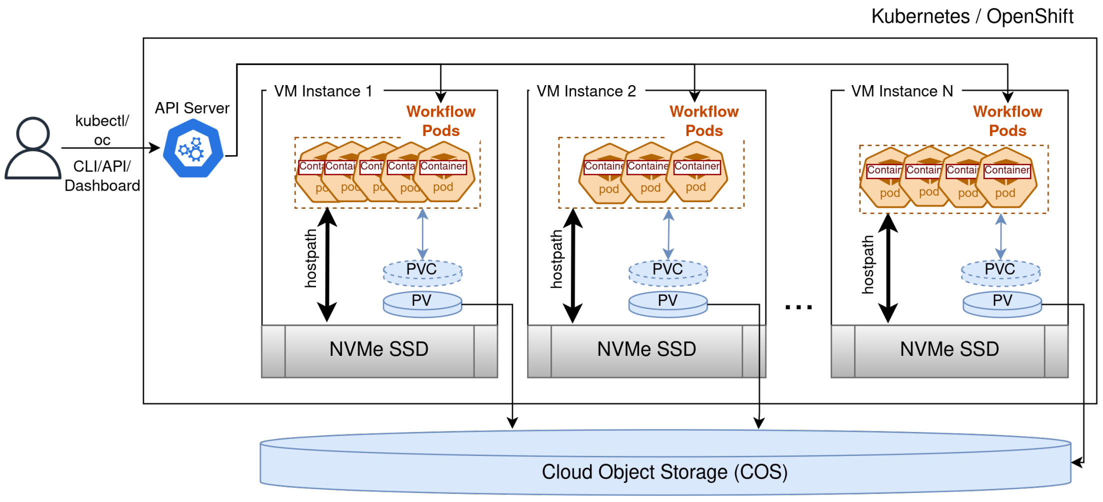


Fig. 2: General Kubernetes and OpenShift cluster architecture.

another. The cluster consists of a set of virtual machine (VM) instances running on top of computational nodes. Since we assume a node can host a single VM instance, we use both words interchangeably across the paper. Each *VM instance* hosts one or more *pods* that are the foundational unit of these container management systems. Each *pod* hosts a workflow task in a *container*. On-disk files in a *container* are ephemeral; therefore, *volumes* can connect storage to the computational resources. Ephemeral volume types have a pod's lifetime, but persistent volumes (PV) exist beyond the pod's lifetime, 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, such as cloud object storage (COS). Another PV is a hostpath that allows mounting a file or directory from the host node's file system into the Pod.

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

III. Node-Local Challenges in the Cloud

Node-local storage eases Tier 1's bandwidth as an SSD buffer between compute and storage systems, reducing I/O demands. However, these performance gains come with data persistence, durability, and shareability trade-offs. The direct connection to computational resources provides low latency and high bandwidth but introduces challenges in ensuring data persistence and shareability in cloud-based scientific workflows.

A. Challenge 1: Data Persistence

When using node-local SSD storage, data is ephemeral with a node life cycle and is limited by the capacity of the node-local SSD. 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, earn 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 cloud infrastructure, 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. Another aspect of Tier 0 storage is the capacity of the devices. Most systems offer node-local SSDs ranging from 100 GB to a maximum and not very common 10 TB. However, for some scientific workflows [16]–[18] 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.

B. Challenge 2: Data Shareability

When using node-local SSD storage, data is not shareable across VM instances but local to the VM instance where the pod is executed. Scientific workflows deploy advanced algorithmic techniques to transform the data and produce scientific discovery. These techniques leverage parallelism to improve performance. For example, scientific workflows require parallel techniques for distributing data and models due to the rapid growth of artificial intelligence (AI) and machine learning (ML) and the significant amounts of data required to train a model and apply inference to new datasets.

In Fig. 3, 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, aggregating and averages to update the weights. When deploying node-local storage, the training data can be fully loaded and cached in a single node (our example) or loaded per batch on each node where it trains. If it is the first case, the NN models executed on different nodes 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

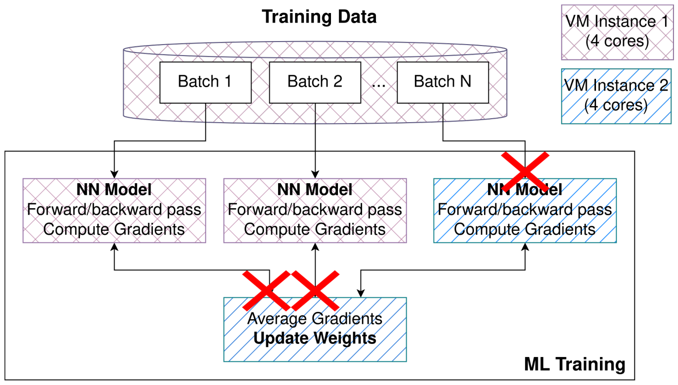


Fig. 3: Demonstration of a failure when deploying a common ML training workflow pipeline using node-local storage due to the lack of shareability across nodes.

workflows, such as 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 executing workflows that deploy parallelism techniques and require high I/O performance.

IV. Persistent and Shareable Data

We design a novel software architecture that efficiently manages data with node-local storage, addressing application and system requirements. At the application level, we automate workflow orchestration without modifying the original code, while at the system level, we ensure data persistence and shareability. We outline the architecture, component interactions, and cloud-based implementation.

A. *Orchestrating Scientific Workflows*

We enable iterative development and automatic orchestration of end-to-end scientific workflows in cloud infrastructure. We leverage the cloud's customization to add functionalities through operators. These operators are application-specific controllers that extend the functionality of 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 the 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 in which the tasks are called in the pipeline flow. We add the data connections (input and output), 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), which enables Infrastructure as Code (IaC). This dynamically translates the pipeline from the domain-specific language into the cloud cluster resources (e.g., pods, PVCs, PVs). For example, each task is translated as a pod. Finally, we obtain a cloud-defined workflow that the pipeline operator deploys in the cluster depending on the available resources.

B. *Persisting Data*

We adopt the concept of burst buffers (e.g., DataWarps, Data Rabbits) widely used in HPC and design an operator called PerSSD (Persistent, Shared, and Scalable Data) that provides similar functionalities for cloud environments. Burst buffers combine software and hardware to use node-local SSDs (Tier 0) as a buffer between compute resources and Tier 1 persistent storage (parallel file systems), thereby improving bandwidth. We adapt this concept for the cloud to leverage the high throughput and low latency of writing and reading to node-local storage. Our operator asynchronously tracks and transfers data from node-local storage to persistent storage (such as cloud object storage). This enables efficient handling of VM migration without requiring special user coding and configuration. The 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 Algorithm 1. PerSSD initiates and runs while a pipeline workflow is executed. During initialization, our operator launches an empty database filled out during the workflow execution to keep track of the pods and identify when to transfer the data. The database stores metadata from the pod and workflow DAG, including the pod's unique ID, name, task, execution node, list of dependent child pods, output data, a flag indicating if the data was transferred to persistent storage, and the timestamp of data transfer.

After initialization, PerSSD runs in a control loop receiving updates of the pods' statuses (*Initializing, Pending, Running, Completed*). When a pod completes without errors, PerSSD

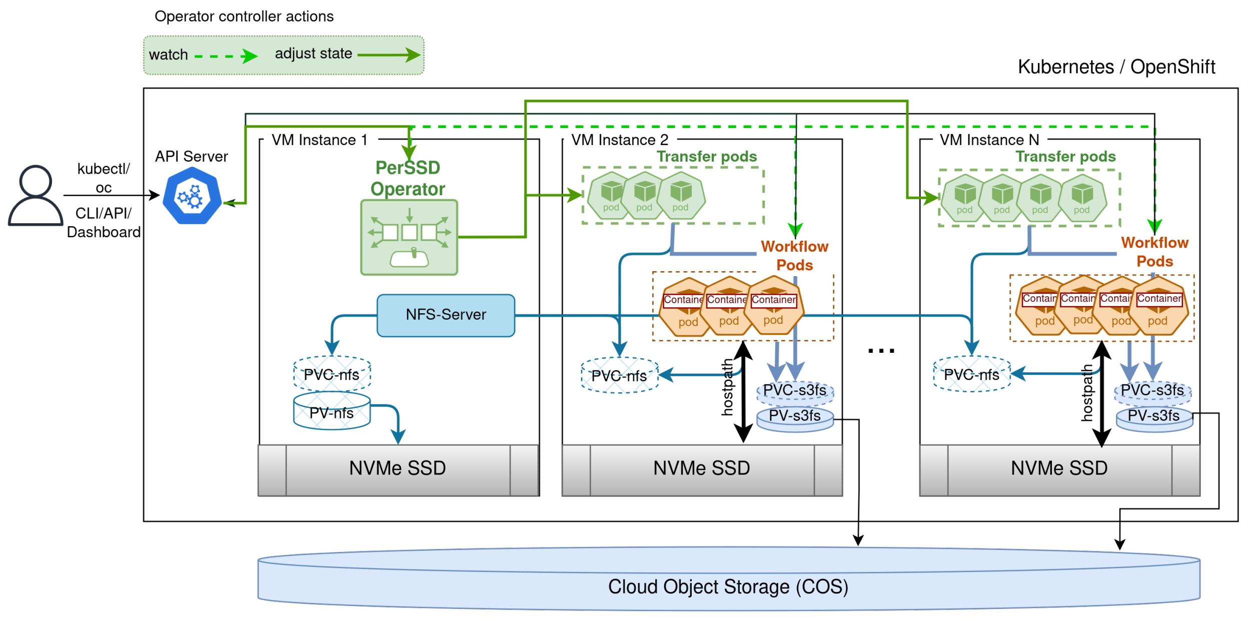


Fig. 4: Architecture of PerSSD with its three components: the workflow pods, the PerSSD operator, and the NFS server on top of the local NVMe SSD. The workflow tasks are orchestrated using Open Data Hub (ODH), Kubeflow, and Tekton Pipelines.

Algorithm 1 PerSSD Controller's Logic

```
1:  if $pipelineworkflow$ starts then
2:      Initialize database
        $db \leftarrow [ID, podname, taskname, node, outputs, children, pushed, time]$
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:                 $db[pod_{ID}][pushed] \leftarrow True$
```

extracts metadata information (pod name, task name, 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 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 completed, the pushed flag in the database is updated to $True$. This logic repeats until the pipeline workflow is completed. 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 and any other tenants on the physical node.

C. 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 set up a Network File System (NFS) on top of the node-local SSDs and connect all pods required to share data across the network. We select NFS because (i) it is a standard part of the Linux kernel, making it widely compatible without special software configuration, (ii) it provides centralized management of data across nodes with easy integration and transparent access to data, and (iii) it enables data-locality awareness, accessing the data across the network of multiple nodes without user intervention. These properties make NFS a widely used file system in the cloud. Other file systems such as BeeGFS [19] and GekkoFS [20] provide cloud operators for distributed access but are currently built on convoluted configurations and unstable releases.

D. 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 Open Data Hub (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 is the shared file system across all workflow pods. Fig. 4 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 (i.e., it manages the components of our architecture). This node hosts the operator and the NFS server; thus, we allocate more resources to this node than the other worker nodes, so the architecture performs as expected.

We use the Open Data Hub Pipeline Operator [21] to generate the workflow pods as part of the pipeline. We select ODH because is intended for cloud infrastructure 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 [22] and Tekton pipelines [23] together to design, manage, track, execute, and view data-driven pipelines. Kubeflow Pipelines 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 cloud resources. We select Tekton as the translation package.

We build our operator using the Kubernetes Operator Pythonic Framework (Kopf) [24]. 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 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.

We design our architecture with resilience by considering workflow and infrastructure failures. Resilience at the workflow level is delegated to the application and the scientist. When a task is incomplete because of workflow failures, our architecture provides detailed logs so scientists can make informed decisions regarding their workflow for the next execution. Our architecture is equipped to automatically address infrastructure failure that can occur during or after the workflow execution. Examples of infrastructure failures during execution include node failure or maintenance, NFS server failure, and out-of-space node-local storage. To address these failures, our orchestration has a `retry` flag where the impacted tasks are relaunched until successful completion. The user defines the `retry` flag recurrence by default. Infrastructure failures after the workflow's execution include a node being de-allocated while transferring the data from node-local to persistent storage. The operator creates a new transfer job to recover data by collecting metadata, such as the node identity. If the node is active, the job completes the transfer; otherwise, the task is relaunched on a different node, regenerating and transferring the data to persistent storage.

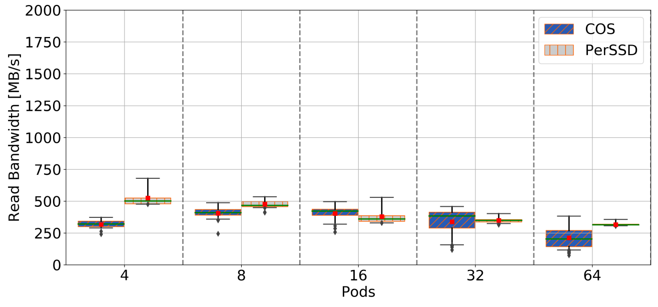
V. Workflow Scalability

We evaluate the I/O performance and execution time of workflows using our PerSSD architecture compared to a Tier 1 setup, focusing on data persistence and shareability. The evaluation includes two scenarios: (i) an FIO benchmark and (ii) a real scientific workflow. We assess PerSSD's scalability and its impact on workflow makespan with effective node-local data management. Tests were conducted on an OpenShift public cloud cluster with 17 nodes, where the control plane node has 32 cores, 128 GB RAM, and a 600 GB NVMe SSD, while the other nodes have eight cores, 32 GB RAM, and 300 GB NVMe SSDs. Each test was repeated five times at different times, with performance variability displayed using boxplots.

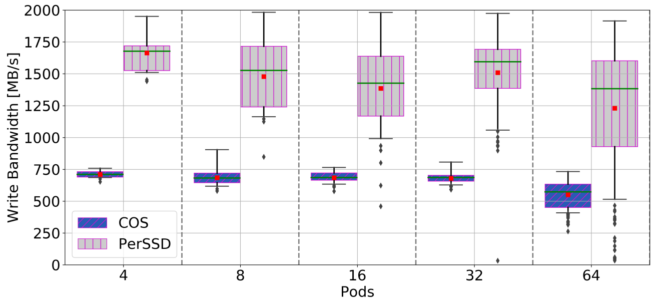
A. *FIO Benchmark*

We demonstrate the generality of our approach by testing on a simulated workflow using FIO [25]. Fig. 6 shows the workflow structure 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 data size 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 [26] 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 training 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 [27], the total data is 150 GB, and models trained on this dataset vary from 1.03 M to 2440 M parameters (4.12 GB to 9.8 GB in float32).

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 Fig. 5). We observe that the write bandwidth, shown in Fig. 5b, outperforms the read bandwidth in Fig. 5a for both storage configurations. The NFS server utilizing top node-local



(a) Read Bandwidth.



(b) Write Bandwidth

Fig. 5: I/O bandwidth scalability comparison between Cloud Object Storage (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.

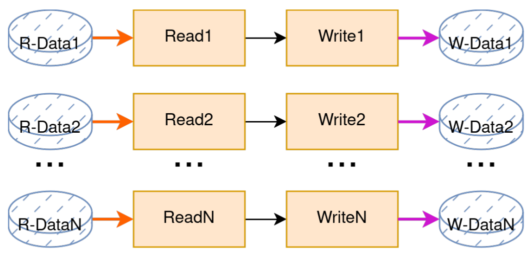


Fig. 6: FIO workflow structure with N tasks in parallel that read N datasets, each with a fixed size, followed by N writing tasks to N independent files.

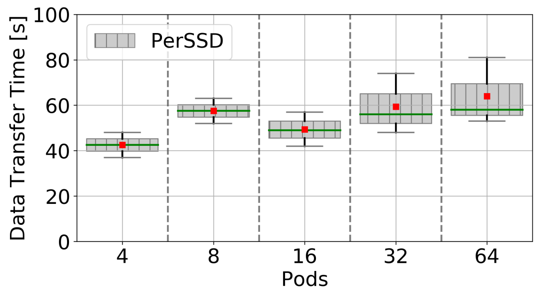


Fig. 7: Data transfer time that takes the operator to transfer all the data to COS after the workflow completes when running the FIO benchmark for different pod configurations.

storage achieves the highest write bandwidth. We also observe in Fig. 5a 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 Fig. 5b. As we increase the number of pods, we notice more variability as we make more requests to the server, which affects performance. Exploring the NFS server's sweet spot of performance based on parallel requests is outside the scope of this paper. With the NFS server ingress bandwidth exceeding the combined data sizes, the node local PerSSD approach offers superior performance without sacrificing the data safety from persistent media. NFS makes moving a VM to another node and accessing data on the previous node invisible. The NFS software knows where the data is stored and can access it across the network without user intervention.

We quantify the time from when the workflow ends to the complete data transfer to COS. Fig. 7 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; thus, at the end of the workflow, we are not transferring all $N \times 3.2GB$ of data.

B. Earth Science Workflow

We apply our approach to an earth science workflow [16] that 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 [28], [29]. This workflow is data-driven and has four components: data generation, data transformation, ML modeling, and analysis. We focus on the data generation stage [30] 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 Fig. 8. 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 calculated, the workflow merges and reprojects the high-resolution TPs.

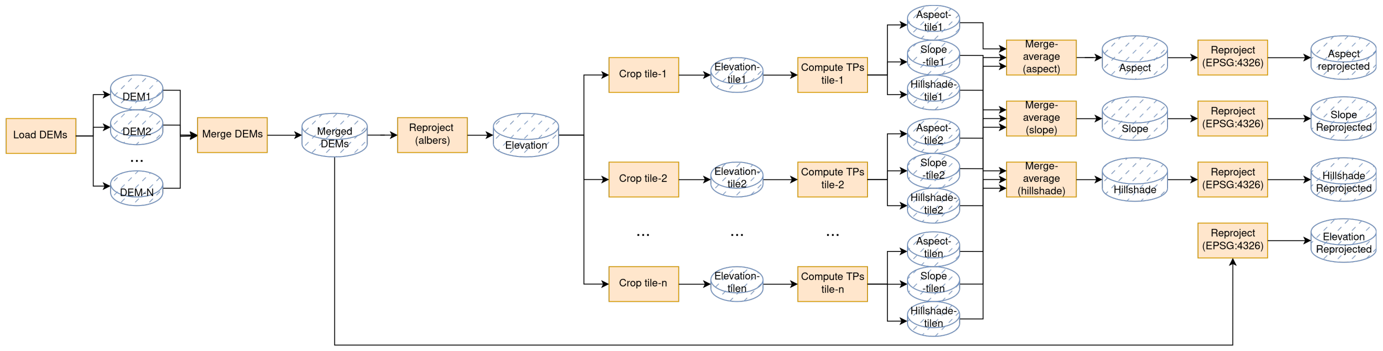


Fig. 8: Earth science workflow that leverages data parallelism for the generation of high-resolution terrain parameters (aspect, slope, hillshade) from Digital Elevation Models.

TABLE I: 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

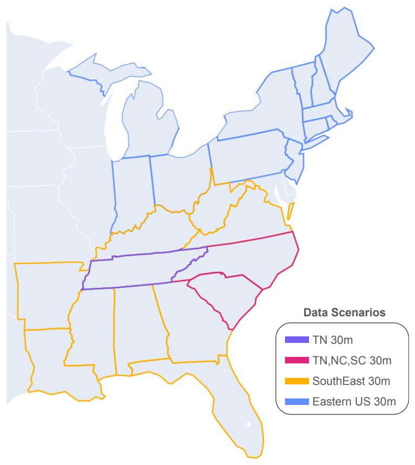


Fig. 9: Data scenarios for scalability: (i) Tennessee (purple), (ii) three southeastern states: TN, NC, SC (red), (iii) all southeastern states (yellow), and (iv) the eastern USA (blue).

We measure the overall makespan of the workflow in Fig. 8 as we scale out from a state to the eastern USA (Fig. 9). We generate TPs for four regions at a 30 m resolution and Table I presents 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 Fig. 10, we visualize the total time 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. Fig. 11 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 10 and 11 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.

VI. Related Work

Our PerSSD architecture is an integrated solution between the application and the system where the scientific workflow orchestration directs the data management and storage. We present state-of-the-art practices and tools for orchestrating scientific workflow and show purely storage software platforms that facilitate the usage of node-local storage.

Scientific Workflow Orchestration Traditional workflow management systems (WMs) orchestrate workflows on heterogeneous resources but are often complex and lack portability [3], [31], [32]. Container orchestration addresses this by isolating components as standalone units [10], [33], [34], with systems like Pegasus, Nextflow, and Galaxy integrating these features [4], [5], [35]. However, HPC-based WMs are costly [36]–[38], and cloud solutions face vendor lock-in [39]–[41]. Our cloud-native solution expands Kubernetes orchestration via a pipeline operator, offering open-source, vendor-independent deployment without code modification.

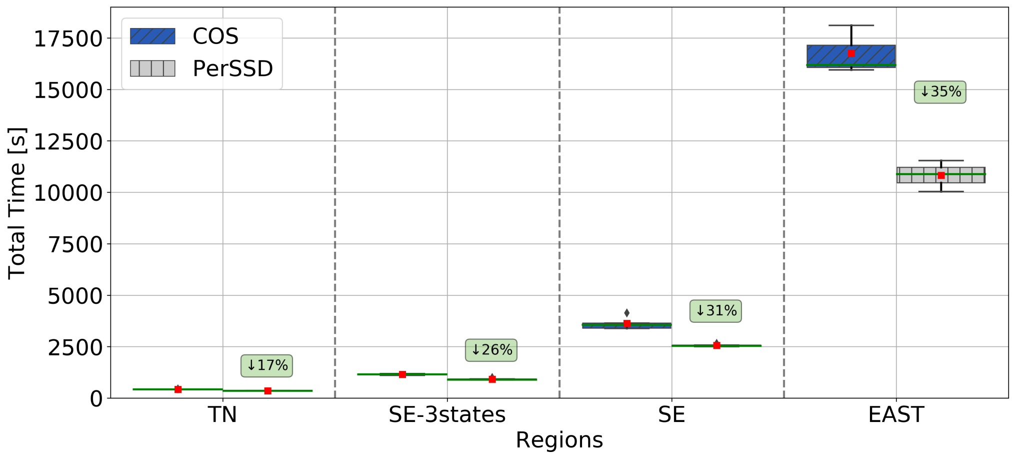


Fig. 10: Total time comparison between executing the workflow deploying COS and using node-local storage with PerSSD. The total time includes the time from the start of the workflow until the allocation of all data in persistent storage.

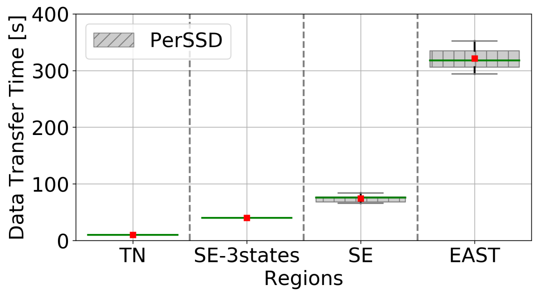


Fig. 11: Data transfer time for the four scenarios.

Node-local Management: Node-local storage has been explored in HPC for two-tiered architectures and burst buffers [42], [43], but cloud environments rely on external object stores [1], [44], [45]. Solutions like OpenEBS focus on resilience [46], but our work emphasizes reproducibility by integrating data annotation and workflow orchestration with storage management. PerSSD ensures persistence and shareability in cloud environments, offering a cloud-native alternative to HPC burst buffers with improved I/O performance using Kubernetes and OpenShift.

VII. Conclusions

This paper presents PerSSD, a novel software architecture that orchestrates scientific workflows using node-local storage while ensuring data persistence and shareability in cloud environments. Our system integrates (i) a pipeline operator for workflow automation, (ii) an operator for ensuring data persistence, and (iii) a file system for sharing data across nodes. We demonstrate improved performance in two scenarios: (i) an FIO benchmark, showing a 50% increase in write bandwidth compared to COS, and (ii) an earth science workflow, reducing the makespan by 35%. As data scales, PerSSD shows greater performance gains over COS. The time to transfer data to COS asynchronously is minimal (0.33%), making total execution time faster than COS alone. Future work will explore parallel systems for node-local storage and testing across different cloud vendors.

VIII. Code Availability

We provide the code for our PerSSD architecture, including the PerSSD operator, NFS server, and the orchestration implementation for FIO and the earth science workflow at *https://github.com/TauferLab/perssd-nvme/tree/main*.

Acknowledgments

The authors acknowledge the support of IBM through a Shared University Research Award; Sandia National Laboratories; the National Science Foundation through the awards #1900888, #2331152, #2138811, #2103845, #2223704, and #2334945; and the ACCESS program through the NSF grant #2138296.

References

[1] V. Bucur, C. Dehelean, and L. Miclea, "Object Storage in the Cloud and Multi-cloud: State of the Art and the Research Challenges," pp. 1–6, 2018.

[2] F. Di Natale, H. Bhatia, T. S. Carpenter *et al.*, "A massively parallel infrastructure for adaptive multiscale simulations: modeling ras initiation pathway for cancer," in *Proc. of the International Conference for High-Performance Computing, Networking, Storage and Analysis*, ser. SC '19, 2019.

[3] 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)*, 2016, pp. 1–8.

[4] 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)*, 2019, pp. 281–290.

[5] 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, no. 4, pp. 316–319, Apr 2017.

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

[7] 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)*, 2020, pp. 1027–1035.

[8] 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.

[9] 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 *Proc. of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*, 2024.

[10] 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)*, 2023, pp. 383–393.

[11] 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.

[12] 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.

[13] 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)*. IEEE, 2022, pp. 403–404.

[14] 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*, 2021, pp. 452–465.

[15] 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.

[16] 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)*. IEEE, 2019, pp. 1–10.

[17] 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*, ser. ICPP ’23. Association for Computing Machinery, 2023, p. 1.

[18] P. Olaya, S. Caíno-Lores, V. Lama *et al.*, “Identifying Structural Properties of Proteins from X-ray Free Electron Laser Diffraction Patterns,” in *2022 IEEE 18th International Conference on e-Science (e-Science)*, 2022, pp. 21–31.

[19] “Beegfs,” hwww.beegfs.io/c/, [Online; accessed 05-08-2024].

[20] 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)*, 2018, pp. 319–324.

[21] “Open data hub plaform,” https://github.com/opendatahub-io/data-science-pipelines-operator , [Online; accessed 05-08-2024].

[22] “Kubeflow,” https://github.com/kubeflow/kubeflow, [Online; accessed 09-25-2023].

[23] “Tekton pipelines,” https://github.com/tektoncd/pipeline, [Online; accessed 09-29-2023].

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

[25] J. Axboe, “fio: Flexible I/O ,” available at https://github.com/axboe/fio, version 3.33 [Online; accessed 02-25-2023].

[26] 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*, 2008, p. 15–24.

[27] “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*. Ieee, 2009, pp. 248–255.

[28] 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.

[29] 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.

[30] 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*, 2023, p. 311–312.

[31] 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)*, 2018, pp. 2773–2775.

[32] L. Wratten, A. Wilm, and J. Göke, “Reproducible, Scalable, and Shareable Analysis Pipelines with Bioinformatics Workflow Managers,” *Nature Methods*, vol. 18, no. 10, pp. 1161–1168, Oct 2021.

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

[34] 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.

[35] 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.

[36] 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*, 2015, pp. 22–27.

[37] 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*, ser. ICPP Workshops ’21. Association for Computing Machinery, 2021.

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

[39] 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.

[40] Y. Hu, H. Wang, and W. Ma, “Intelligent cloud workflow management and scheduling method for big data applications,” *Journal of Cloud Computing*, vol. 9, no. 1, p. 39, Jul 2020.

[41] 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.

[42] 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)*, 2022, pp. 246–255.

[43] 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*, 2017, p. 41–52.

[44] 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.

[45] 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.

[46] “Openebs,” https://github.com/openebs, [Online; accessed 05-08-2024].