Orchestration of Materials Science Workflows for Heterogeneous Resources at Large Scale

Journal Title
XX(X):1–10

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Naweiluo Zhou[1], Giorgio Scorzelli[2], Jakob Luettgau[1] Rahul Reddy Kancharla[3] Joshua J. Kane[3] Robert Wheeler[4] Brendan P. Croom[5] Pania Newell[2] Valerio Pascucci[2] and Michela Taufer[1]

Abstract

In the era of big data, materials science workflows need to handle large-scale data distribution, storage, and computation. Any of these areas can become a performance bottleneck. We present a framework for analyzing internal material structures (e.g., cracks) to mitigate these bottlenecks. We demonstrate the effectiveness of our framework for a workflow performing synchrotron X-ray computed tomography reconstruction and segmentation of a silica-based structure. Our framework provides a cloud-based, cutting-edge solution to challenges such as growing intermediate and output data and heavy resource demands during image reconstruction and segmentation. Specifically, our framework efficiently manages data storage, scaling up compute resources on the cloud. The multi-layer software structure of our framework includes three layers. A top layer uses Jupyter notebooks and serves as the user interface. A middle layer uses Ansible for resource deployment and managing the execution environment. A low layer is dedicated to resource management and provides resource management and job scheduling on heterogeneous nodes (i.e., GPU and CPU). At the core of this layer, Kubernetes supports resource management, and Dask enables large-scale job scheduling for heterogeneous resources. The broader impact of our work is four-fold: through our framework, we hide the complexity of the cloud's software stack to the user who otherwise is required to have expertise in cloud technologies; we manage job scheduling efficiently and in a scalable manner; we enable resource elasticity and workflow orchestration at a large scale; and we facilitate moving the study of nonporous structures, which has wide applications in engineering and scientific fields, to the cloud. While we demonstrate the capability of our framework for a specific materials science application, it can be adapted for other applications and domains because of its modular, multi-layer architecture.

Keywords

Scientific workflow, orchestration, resource management, job scheduling, materials science, cloud computing.

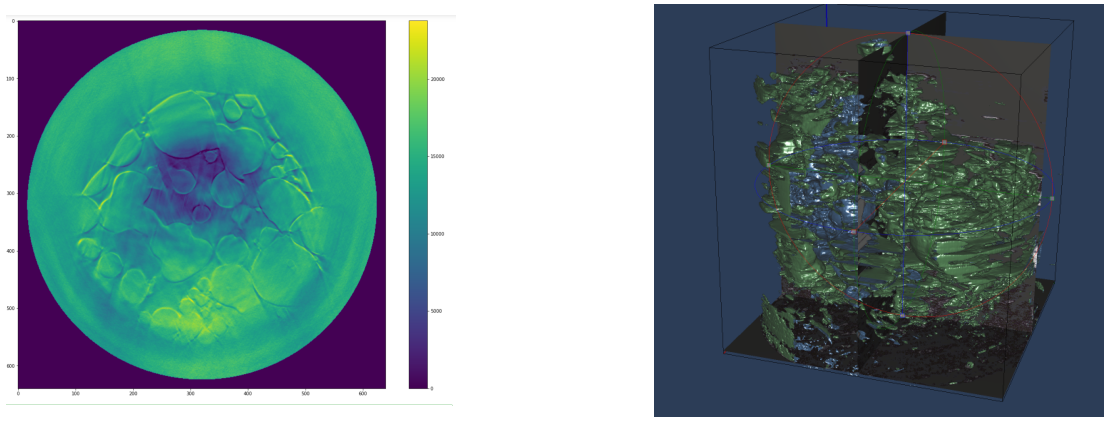
Motivation and Proposed Solution

Materials science has evolved to adopt sophisticated scientific workflows to automate experiments and data analysis. Materials science workflows can express many applications, often consisting of multiple tasks that require resources beyond a single machine Wu et al. (2015). A workflow usually comprises several stages, executing a task and taking input data from experimental tools or previous steps. Materials science workflows can generate and stream massive amounts of data in the order of terabytes or petabytes. Examples of these workflows include analyzing and visualizing internal material structure by performing synchrotron X-ray computed tomography reconstruction and segmentation of silica. A personal computer or a small dedicated cluster is no longer sufficient to perform the reconstruction and segmentation operations on tomography data. Public cloud platforms can effectively provide computing and storage resources, as users do not need to deploy an entire infrastructure that brings additional costs such as hardware purchase and maintenance Cepuc et al. (2020). Furthermore, cloud resources are provisioned based on user demands with a pay-as-you-go model and can quickly adapt program execution environments to user needs. Still, efficiently deploying services on the cloud requires technical expertise.

In this work, we break through the expertise barrier by providing scientists in materials science with a framework that enables a scalable, easy use of the cloud for their studies. We leverage cutting-edge cloud technologies to design and implement our framework. To this end, our framework integrates a rich suite of software features needed for managing the heterogeneous hardware infrastructure (CPUs, GPUs, and storage) on the cloud and orchestrating the tasks composing the multi-stage materials science workflow, removing the need for a deep understanding of the cloud infrastructure or deploying workflow management systems Deelman et al. (2015); Jain et al. (2015); Wilde et al. (2011). Furthermore, our framework enables interactive, visual exploration of large materials science data. Our

[1] University of Tennessee, Knoxville, USA
[2] University of Utah, Salt Lake City, Utah, USA
[3] Idaho National Laboratory, Idaho Falls, USA
[4] MicroTesting Solutions LLC, Hilliard, Ohio, USA
[5] Johns Hopkins University Applied Physics Laboratory, Laurel, USA

Corresponding author:
Michela Taufer, taufer@utk.edu



(a) 2D rendering

(b) 3D rendering

Figure 1. A single slice of the materials in the z-axis view after reconstruction in 2D rendering (a) and the image with enhanced contrast and removal of artifacts after segmentation viewed in 3D rendering (b). Note that the images at both stages can be 3D rendered.

framework comprises three layers: a top user interface, a middle layer for resource deployment, and a lower layer dedicated to resource management.

Using synchrotron X-ray computed tomography, we deploy our framework to study *in situ* mechanical testing of a nanoporous structure. Such study has various applications in engineering and scientific domains (from fillers and additives to adsorbents, catalysts, and recently therapeutic agents and vaccines in nanomedicine). The workflow studying the phenomena comprises six stages: downloading images from X-ray tomography scans (6.0 GB per image); performing image segmentation to remove noises; performing image reconstruction to obtain a clear vision of the material structure; converting the data into OpenVisus formats Petruzza et al. (2020); streaming data to storage; and using the OpenVisus streaming services for user analysis. Figure 1 shows an example of the internal structure of a porous silica material at two different stages. Figure 1.a shows a single slice of the materials in the Z-axis view in 2D rendering after reconstruction. Figure 1.b shows the enhanced contrast and removal of artifacts after segmentation in 3D rendering.

We show how our framework provides a scalable solution to the study of the internal structure of a porous silica material in which an increase in data size requires storage resource elasticity provided by the cloud technologies building our framework. The contributions of this paper are as follows:

- We present our multi-layer framework that provides a complete life cycle for a materials science workflow from its environment provision to final stage data analysis using cutting-edge cloud technologies.
- We use our framework to study the *in situ* mechanical testing of a nanoporous structure; nonporous structures have wide applications in engineering and scientific fields.
- We demonstrate the ability of our framework to handle large amounts of data at a magnitude of ten TBytes in a scalable manner using a public cloud platform such as Chameleon Keahey et al. (2020).

Although the framework capabilities are demonstrated for a specific study in the materials science domain, it can be easily adapted to support multi-stage, modular workflows in other domains Singh et al. (2019). The modular implementation of our framework allows domain scientists to easily plug in their software modules as new workflow stages and run their workflow with the framework's Jupyter notebook interface.

Critical Problem in Materials Science

Using synchrotron X-ray computed tomography, we leverage our framework to study a critical data and computation-intensive problem in materials science: the *in situ* mechanical testing of a nanoporous foam. Porous silica-based materials have wide applications in engineering and scientific domains, from fillers and additives to adsorbents, catalysts, and recently therapeutic agents and vaccines in nanomedicine Lei et al. (2020). The internal structure of porous silica-based materials can be reconstructed from synchrotron X-ray tomography. Tomography combines the projections of a sample slice at different orientations taken by placing the slice between an X-ray source and detector Maire et al. (2001). Analysis of the *in situ* tomography images can reveal the underlying mechanisms that drive macroscopic mechanical performance Peukert et al. (2015), such as the collapse of pores, cracking and buckling of ligaments, and enable the design of higher-performing silica foams.

The tomography images obtained from X-ray scans require image processing to perform synchrotron X-ray computed tomography reconstruction and segmentation of silica. Reconstruction and segmentation remove blurriness, calculate the tomographic center of rotation, and eliminate scanning artifacts. Figure 2 illustrates the complete experiment process represented in a scientific workflow. The method comprises six stages performed on a sequence of images: downloading the images of X-ray tomography (Stage 1); performing the image reconstruction to obtain a clear vision of the materials structures (Stage 2); performing the image segmentation to remove background noise, enhance contrast, and reduce image size (Stage 3); converting the data into a network-friendly multi-level

format (Stage 4); streaming data to users (Stage 5); and visualizing and analyzing data with a Jupyter notebook (Stage 6) Randles et al. (2017). Specifically, each X-ray tomography image is approximately 6.0 GB; the images are downloaded in HDF5 file Group (2000-2010). The stages of performing image reconstruction and segmentation (i.e., Stages 2 and 3) are time-consuming, requiring massive computing resources. Intermediate stages (i.e., Stages 2, 3, and 4) yield a large amount of intermediate data reaching at least 30 times the input image size. Figure 2 shows examples of intermediate images after the segmentation and reconstruction stages (images on the left). Preserving the intermediate data generated by each stage facilitates scientists to track data provenance Wang et al. (2015), enabling the reusability of workflow building blocks and the reproducibility and trustworthiness of the results Taufer et al. (2021a,b); Stodden et al. (2016).

Conceptual Design of our Framework

We need a software solution that works on cloud platforms to address the challenges of materials science workflows. Our software solution combines workflow management and resource provisioning with the concurrent execution of the workflow stages using the task parallelism paradigm. Data is streamed with the support of orchestration services from the object storage to the user interface in an interoperable, portable, and general manner.

Software Solution and Hardware Infrastructures

Our software solution is a multi-layer framework composed of three layers: a top user interface, a middle layer for resource deployment, and a lower layer dedicated to resource management. Figure 3 illustrates the multi-layer organization of our framework.

Jupyter is a free, open-source, interactive tool that researchers can use to combine software code, computational output, explanatory text, and multimedia resources in a single document. A Jupyter notebook interface serves as the top layer and facilitates the domain scientist's interaction with the workflows, the interpretation of its execution, and the explainability of its results. We developed ad-hoc notebooks that leverage the OpenVisus Software Development Kit (SDK) Pascucci et al. (2012) to:

- visualize framework outputs (i.e., 2-D and 3-D images);
- perform fast visualization of massive data; and
- run visualization services agnostic from the underneath hardware.

The middle layer uses the declarative markup language YAML to configure, describe, and run all the workflow stages modular and adaptable, making new stages easily identifiable and pluggable into the original workflow. For the setup, we run our configuration scripts on the Ansible Sammons (2016) platform, an IT automation system that handles configuration management, application deployment, cloud provisioning, ad-hoc task execution, and Infrastructure-As-Code (IaC) through a suite of Python modules. For the material science part, we develop a domain-specific and modern software stack that provides services for high-level task definition; low-latency task coordination and communications (our dependencies are expressed as a parameterized Directed Acyclic Graph); automatic failure handling; and automatic data caching, central logging, observability tools.

The lower layer manages heterogeneous computing resources (i.e., GPU and CPU nodes) and schedules different jobs on these resources. The core of this layer uses Kubernetes and Dask: Kubernetes Hightower et al. (2017) is an open-source system for automating deployment, scaling, and management of containerized applications; and Dask Rocklin (2015) is a flexible open-source Python library for parallel computing, scaling Python code from multi-core local machines to large distributed clusters or large-scale distributed cloud platform. For this layer, we intentionally architecture a hybrid resource management system (see Figure 4) where Dask performs large-scale job scheduling and resource management for GPU-intensive tasks, and Kubernetes orchestrates long-running CPU-based services such as the data-streaming services. We leverage three types of gateways (also known as entry points or control nodes) that provide indirect, monitored, and secured access:

- an Ansible Control Node orchestrates and performs cluster-wide deployments;
- a Kubernetes Login Node dispatches the commands to the Pods (i.e., atomic unit on the Kubernetes platform); and
- a Jupyter Serve Node launches on-demand multi-user Jupyter notebooks allowing interactive analysis.

The storage layer at the bottom of our software stack includes temporary local file storage and remote object storage. The local file system is used to download locally remote artifacts, such as the HDF5 files produced by the acquisition devices, temporary store artifacts produced by the reconstruction and segmentation algorithms, and support the artifacts conversion to Analysys-Ready Network Optimized (ARCO) file formats Abernathey et al. (2021). Since most of the mentioned operations require random file access and low latency, we need a small amount of fast local storage in order of hundreds or thousands of Input Output operations per second (IOPS). The remote object storage, leveraging the S3 Application Programming Interface (API), allows the permanent storage of the results and their distribution and replication at the national level. Our framework is designed from the ground base to be compatible with any commercial or educational object storage solution. We chose Wasabi for this particular workflow, a vendor with a transparent policy of zero-EGRESS fees (i.e., zero cost for moving data outside) and zero API fees. Using Wasabi, we pay for the pure storage costs without incurring high and difficult-to-predict upload and download charges Penny (2020).

Workflow Management and Resource Provisioning

To automate the deployment of our software stack, our framework solution uses Ansible Sammons (2016) that serves as an orchestration tool by offering an easy way to perform cluster-wide operations, such as configuration management, application deployment, cloud provisioning,

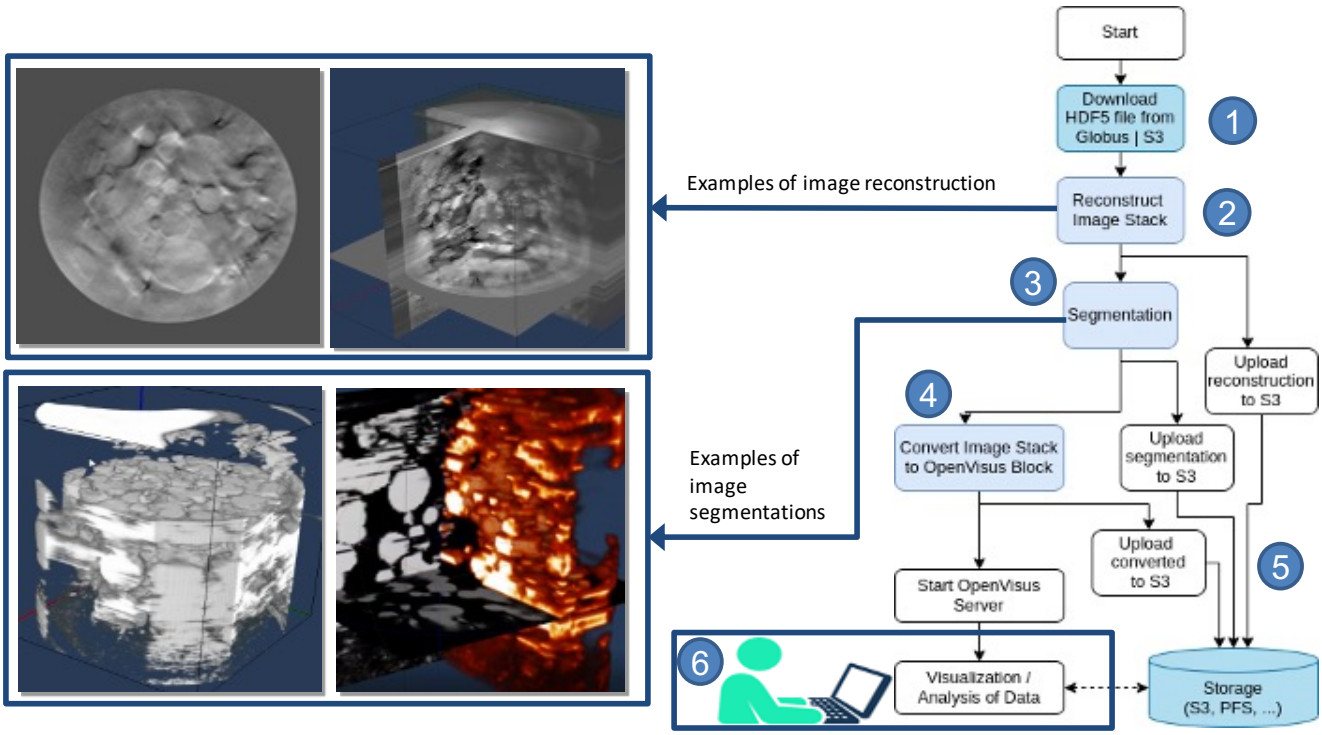


Figure 2. The six stages of a materials science workflow for *in situ* mechanical testing of a nanoporous structure using synchrotron X-ray computed tomography. Numbered circles identify the six stages. Examples of intermediate images from the reconstruction and segmentation stages are on the figure's left.

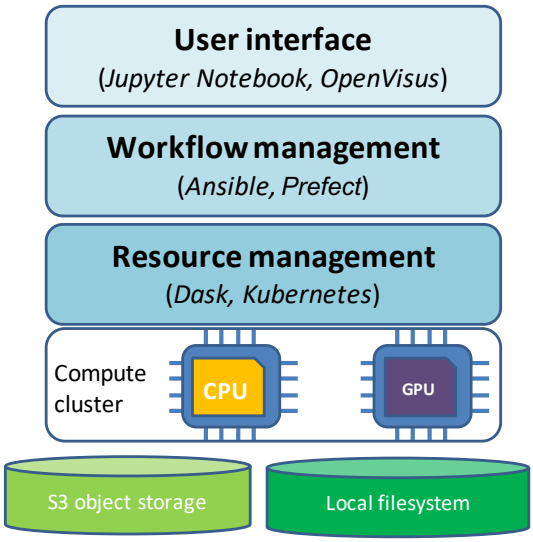


Figure 3. The software layers of our framework on top of the hardware infrastructure.

ad-hoc task execution, and multi-node orchestration. Ansible deploys thereof via its `playbook` written in `yaml` that describes the automation of jobs or tasks. `yaml`* en Kiki et al. (2009) is a high-level human-readable language in configuration files. Kubernetes also utilizes `yaml` scripts to perform the deployments. Connections to nodes in Ansible are via `ssh`, and nodes managed by Ansible are grouped into an *inventory* file defined by users.

Ansible, our *yaml* scripts, and our Python scripts are essential in workflow executions. Ansible performs cluster-wide software deployments such as:

- Provisioning execution environments by installing and switching the list of python libraries used at different stages of the workflow;
- Deploying the Dask and Kubernetes software stacks.

Our `yaml` script defines all the configurations for our workflow (Figure 5). The script has these key steps:

- Give the local location to store results as shown in Line 2 of Figure 5, and the remote location on `s3` to archive all the data as shown in Line 4. The link in Line 4 gives the bucket name of our object storage Factor et al. (2005) on `s3`.
- Set the environment variables such as the Dask worker directory (Line 11). The environment settings are applied for all the Dask nodes.
- Define the stages of our workflow that scientists execute on the cloud. As an example, in our script shown in Figure 5, scientists insert the names of Stages 2, 3, and 4 representing the Python module names. Note that Stages 2 and 3 in this script are incorporated under one component `Preprocess`, as both stages perform heavy computation. The stages are scheduled to run on the Dask cluster. `Convert` represents Stage 4 in Figure 2. In this way, scientists only need to bring their Python modules that focus on the scientific functionalities, and our framework handles the rest (e.g., job scheduling or task parallelization); and
- Specify the series of input files. Line 21 gives the location of an input image filename.h5 on `s3`.

*yaml: yet another markup language.

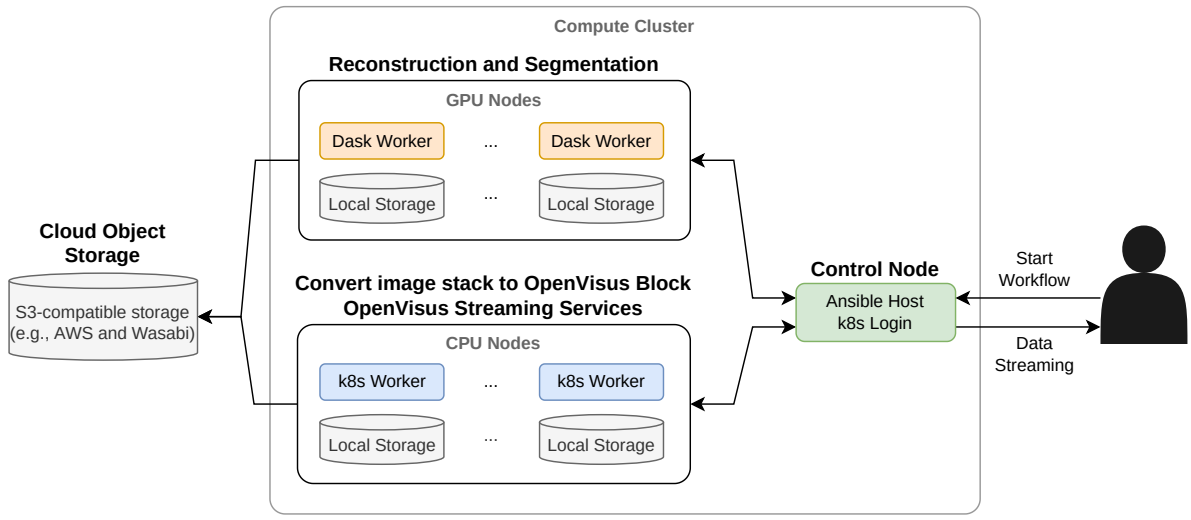


Figure 4. Architecture of our framework. The number of nodes in the figure is only for illustration.

Our Python scripts pass the configurations defined in the `yaml` script to the layers of our framework.

```
1 # where to store local files
2 LOCAL: /tmp/in_situ_data
3 # remote address where to store results
4 REMOTE: s3://in_situ_data
5
6 #ansible inventory location
7 inventory: /ansible/inventory.ini
8 #the list of the hosts defined in inventory.ini
9 group: dask_node_list
10 #dask worker directory
11 worker-local-dir: /tmp/nsdf/dask-workers
12
13 # what task to execute (preprocess | convert)
14 task: preprocess
15
16 # specify which stage of the workflow to execute
17 disable-reconstruction: true
18 disable-segmentation: false
19
20 #specify which file on s3 to download
21 - url: s3://in_situ_data/filename.h5
```

Figure 5. A snippet of the `yaml` script setting configurations for the workflow. The `preprocess` includes two stages: reconstruction (Line 17) and segmentation (Line 18). For simplicity, only a part of the code is shown herein.

Task Parallelization

To execute workflows effectively, we introduce task parallelism in our framework. Figure 6 shows a simplified snapshot of the parallelization. The unit of parallelism is a stage of the workflow as shown in Figure 2; each unit is identified by the standard Python decorator `@task` (Line 1, Line 4, and Line 7). The defined tasks are formed into a pipeline (represented as a *Prefect Flow*) scheduled to GPU nodes by Dask. We use Prefect[†] as our workflow management system on top of Dask. Dask handles scheduling and resource management of `tasks` within a workflow. The Dask cluster consists of one scheduler and many worker nodes. The first node defined in the Ansible inventory is elected as the scheduler and the rest act as workers. A defined `task` is executed in parallel on all the worker nodes. Additionally, our framework records logs of task status on each Dask worker and passes them to the Dask scheduler during execution. The logs can be disabled to reduce communication costs between nodes.

```
1 @task
2 def Reconstruction(d):
3 return CallReconstruction(d)
4 @task
5 def Segmentation(d):
6 return CallSegmentation(d)
```

Figure 6. A snippet of stage operations with parallelized task indicated by @task decorator. Some parts of the code are omitted for simplicity.

Streaming Service Orchestration

To support streaming services on multiple platforms, the entire software stack of our framework is encapsulated inside a Docker container Merkel (2014), with the data being mounted inside the containers. The software stack comprising the three layers (Figure 3) is a collection of tools that are well-tested and have been used for general purposes: Kubernetes as a container orchestrator, Ansible as a cluster-wide deployment tool, OpenVisus as a data visualization tool. Kubernetes schedules the Docker instances and automatically relaunches them when failures occur. The data is exposed externally via Kubernetes services associated with a global IP address and a port number. Multiple users can access the data via a given link. Figure 7 illustrates how a user can access the dataset `id_1265` inside a Jupyter notebook. The streaming service component in our architecture can be decoupled from the framework and deployed on a more permanent yet smaller cluster that can continuously provide long-running streaming services within the budget of small or median-size research laboratories or institutes.

[†]`https://www.prefect.io/`

```
1 # import Openvisus package
2 import OpenVisus as ov
3 # obtain the image named id_1265
4 im =ov.LoadDataset(
  'http://xx.xx.xx.xx:32439/mod_visus?dataset=id_1265')
```

Figure 7. A code snippet on how to fetch the streamed dataset in a Jupyter notebook. The real IP address has been replaced by xx.xx.xx.xx, and the port number, in this case, is 32439.

Local Filesystem and Remote Object Storage

To manage the large volume of intermediate data, each computing node on the Dask and Kubernetes clusters has local storage (i.e., local filesystem) attached. The cloud object store `s3` is accessible to all nodes on the compute cluster as illustrated in Figure 4. Data accessing to `s3` is much slower than the local store. Hence s3 only serves as a complementary data store. The local filesystem enables quick data access during workflow execution. However, it has limited storage capacity and can only provide temporal storage. The hybrid data store allows our framework to handle a large volume of data during workflow execution and also offers long-term storage for data retrieval used by the Kubernetes streaming services.

The input HDF5 files are downloaded from `s3` as shown on the left side of Figure 4 to the local disk. The desired images to be streamed are also fetched from `s3`. The intermediate data generated during execution on the local filesystem is freed once backed up to `s3`, performed in the background by multiple threads during computation.

Interoperability, Portability, and Generalizability

To sustain interoperable workflow executions, our software framework includes a Jupyter notebook as a user interface. Furthermore, the deployment layer provides portability, enabling our framework's application to run on different cloud environments. Last, the various parts of our software architecture are highly generalizable. The workflow configurations are defined in a single file written in `yaml` that users can easily customize without expertise in cloud technologies for specific use cases. In addition, our framework does not modify users' Python modules (*e.g.* the module performing image reconstruction); instead, we plug the module into our framework, which automatically manages parallelization, job scheduling, and resource management.

Performance Evaluation and Scalability

Materials scientists seek workflow execution speed-up and efficient management of the massive amount of generated data. We assess whether our framework meets these goals for our materials science workflow. To this end, we show the scalability of our framework by evaluating the performance of the two most computation-intensive stages (i.e., reconstruction and segmentation) with an increasing number of GPU nodes. We present the increase in intermediate data size at the end of each stage to demonstrate the importance of storage resource elasticity.

We run our workflow on Chameleon Keahey et al. (2020), a public cloud platform supported by the National Science Foundation with sites at the University of Chicago and the Texas Advanced Computing Center. Chameleon is built on the OpenStack cloud software platform. It provides researchers access to nodes of traditional KVM cloud architectures and reconfigurable bare-metal nodes provisioned via OpenStack Ironic. Table 1 gives the Chameleon's hardware specifications of the 16 GPU nodes and three CPU nodes that consist of the Dask and Kubernetes clusters used for our testing, as shown in Figure 4. Figure 8.a presents our scalability study regarding the workflow's total execution time (in hours). The figure illustrates the average time of three executions with 2, 4, 8, and 16 GPU nodes. We can observe how the execution time is significantly shortened by 88.7% running with 16 nodes compared to the two nodes. Table 2 gives the measurements' variation and standard deviation, indicating that the execution time is stable on Chameleon.

The inputs in Stage 1 are 16 HDF5 files, with each having size of 6 GB. Computation is delayed until the files are downloaded to the local disk at each GPU node. The impact of file uploading to `S3` is insignificant since it is performed by background threads when computation occurs. Figure 8.b shows the average download time spent on each GPU node. Our experiments have experienced a stable network as given by the variance and standard deviation in Table 3. However, the network speed can fluctuate depending on cloud providers. For instance, the download speed can become ten times slower than the network's peak speed. Notice that running with two nodes, the variance, and standard deviation are much higher than the rest. The experiment is more likely to undergo network speed variation on the cloud when its execution expands across several days. Consequently, the overall execution time of the two stages is significantly influenced. Nevertheless, compared with the total execution time in our experiment, such cost is still negligible. Table 3 gives the ratio of time cost in file downloading concerning the total time, which is within 1%.

Table 4 gives the data generated stage by stage cumulatively. The volume of streamed images at the end stage reaches approximately ten times the input size. The experiment in this section only evaluates processing time for a small number of tomography of porous silica-based materials with around 100GB, yet amounts to the size at a magnitude of terabyte at the final stage.

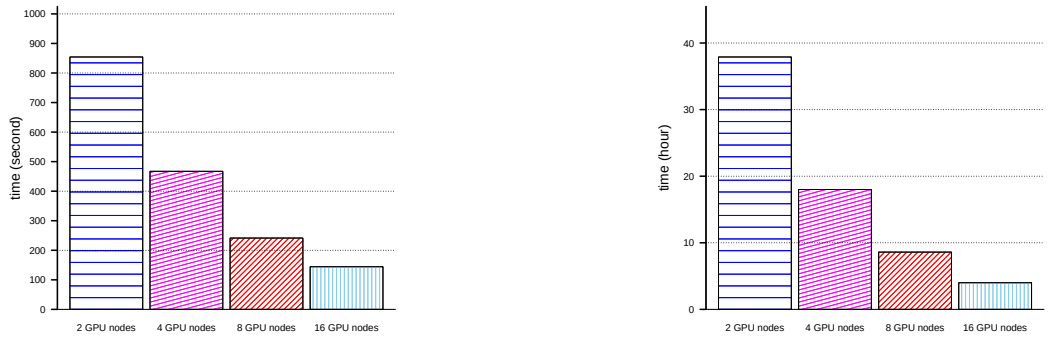
Figure 9 visualizes a set of output images generated with our framework and visualized with our Jupyter Notebook interface (the top layer of our framework). Specifically, the figure shows silica-based nanopillar images generated with synchrotron-based x-ray computed microtomography during a compression test. The novelty aspect of this set of images is the first-of-its-kind visualization of the spherical silica nanoparticles (white to gray color) and void space (black color) in this type of testing.

Aspects of Novelty and Future Directions

Our work is part of a broader effort to port scientific applications to the cloud. We present here a suite of other contributions, outlining for each what our framework introduces the novelty aspects. We also define the future direction to support the reusability, portability, and

Table 1. Hardware specifications of the cloud infrastructures used for scalability evaluation. The Dask and Kubernetes clusters used for our experiments comprise three CPU and 16 GPU nodes.

CPU node	Num. cores	Memory size	CPU vendor	Local storage
	24	187.0GB	Intel Xeon Gold 6126, 2.60GHz	441GB
GPU node	Num. GPU	Memory size	GPU vendor	Local storage
	1	23.4GB	Quadro RTX 6000	210GB



(a) Total execution time (unit: hour)

(b) Download time (unit: second)

Figure 8. Scalability study of the two computation-intensive stages (i.e., Stages 2 and 3) in terms of workflow's total execution time (in hours) (a) and download time per node (in seconds) (b) on Chameleon.

Table 2. The variance and standard deviation of three workflow executions on a different number of GPU nodes. Unit: hour.

Number of nodes	2 GPU nodes	4 GPU nodes	8 GPU nodes	16 GPU nodes
Variance	0.65	0.04	0.08	0.0
Stand deviation	0.81	0.21	0.29	0.06

Table 3. The variance and standard deviation of the time spent downloading the HDF5 files on each node of three execution and its ratio concerning total time. Unit: second.

Number of nodes	2 GPU nodes	4 GPU nodes	8 GPU nodes	16 GPU nodes
Variance	775.5	2.7	8.8	38.4
Stand deviation	27.8	1.7	3.0	6.2

Table 4. Increase dataset size, stage-by-stage, when processing 16 input images.

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6
Accumulative size (GB)	96.7	939.9	1003.9	1022.3	1022.3	1022.3

reproducibility of results. *Jain et al.* Jain et al. (2015) presents a dynamic workflow system used in materials projects to calculate elastic tensors describing the response of materials to external forces within their elastic limits. This workflow system packs and schedules many small tasks for HPC systems that are otherwise not suitable for HPC queuing systems. *Lingertfelt et al.* Lingerfelt et al. (2016) presented a workflow system BEAM targeted for analysis and modeling of material characterization data on HPC systems. A web portal is given to the user for submitting, monitoring the multi-stage workflow execution, and visualizing the results. The algorithms are adapted to utilize MPI to scale on HPC. Our work provides a lightweight workflow system that enables moving applications to the cloud. The parallel tasks and the big data storage requirements benefit from the cloud Iaas Manvi and Krishna Shyam (2014) (Infrastructure As A Service) model at an economical cost.

Taghizadeh-Popp et al. Taghizadeh-Popp et al. (2020) describes a data-driven platform SciServer developed for astronomy. One key feature of this platform is to perform data filtering, analysis, and visualization close to data in the SQL database. The concept of bringing analysis close to data is fundamental in the era of big data, which is also one of the issues addressed in our architecture, where the inputs are fetched from remote storage to where computation occurs. Differently, our workflow system does not require a central database. Instead, a cloud object storage is used for long-term backup, and the current desired datasets are downloaded to the Kubernetes cluster providing data streaming and visualization services.

Abernathey et al. Abernathey et al. (2021) prototype an architecture named *Pangeo* that is composed of a cloud object storage and compute cluster. The external cloud storage is dedicated to the easy retrieval of large-volume datasets. Kubernetes provision the compute cluster,

Figure 9. Selected output of our framework, generated with the Jupyter notebook interface. The figure shows a first-of-its-kind visualization of the spherical silica nanoparticles (white to gray color) and void space (black color) in silica-based nanopillar images generated with synchrotron-based x-ray computed microtomography during a compression test).

and Dask enables computation parallelism. Differently, our architecture provides an additional deployment layer on top of Dask and Kubernetes, which provisions native environments specified by users for their workflows rather than pulling existing container images, which can be inflexible.

There is a growing need for developing persistent scientific workflows to seamlessly connect and integrate software stacks and data services across cloud platforms supported by virtualization and data provenance Bhatia et al. (2021). Containerization of scientific workflow enables reusability, portability, and reproducibility of results Olaya et al. (2023) and ease of system maintenance efforts Dusia et al. (2015); McDaniel et al. (2015); Monsalve et al. (2015); Herbein et al. (2016). Future directions point to the need for automatic containerization of complete working environments that include software dependencies (e.g., Python programs/modules) at all stages and the components of the Dask cluster, which are currently running natively in our workflow to maximize performance. Containerized Dask components can be deployed to the corresponding GPU nodes through Ansible, automatically generating and scheduling the containers to connected nodes.

Conclusion

The software framework presented herein solves multi-stage workflows in scientific domains. The framework enables automatic workflow scheduling, scalable data management at runtime, and interactive data visualization. Domain scientists can easily plug in their Python modules as the workflow stages and run the workflows with a Jupyter notebook. In addition, this framework also offers a deployment layer to set up the software stacks on cloud platforms and provisions dependencies for programs according to their needs. We described a complex materials science workflow in the context of *in situ* mechanical testing of a nanoporous structure and explored the performance benefits brought by our framework. We presented the technical innovations that can extend the materials science workflow into a generalizable and scalable framework. Future work includes integrating our workflow into data commons, such as the Materials Commons and Digital Rock, which will collect the intermediate data enabling automatic indexing and metadata generation for data searchability.

Supplemental Material

The framework software and the Jupyter Notebook are both open access at https://github.com/nsdf-fabric/nsdf-materialscience.

Acknowledgment

This research was supported by the National Science Foundation (NSF) under grant numbers #1841758, #2028923, #2103845 and #2138811; the Extreme Science and Engineering Discovery Environment (XSEDE) under allocation TG-CIS210128; Chameleon Cloud under allocation CHI-210923; and IBM through a Shared University Research Award. Author BPC was supported by the JHU/APL Independent Research and Development Program.

References

Abernathey RP, Augspurger T, Banihirwe A, Blackmon-Luca CC, Crone TJ, Gentemann CL, Hamman JJ, Henderson N, Lepore C, McCaie TA, Robinson NH and Signell RP (2021) Cloud-Native Repositories for Big Scientific Data. *Computing in Science Engineering* 23(2): 26–35.

Bhatia H, Di Natale F, Moon JY, Zhang X, Chavez JR, Aydin F, Stanley C, Oppelstrup T, Neale C, Schumacher SK, Ahn DH, Herbein S, Carpenter TS, Gnanakaran S, Bremer PT, Glosli JN, Lightstone FC and Ingolfsson HI (2021) Generalizable Coordination of Large Multiscale Workflows: Challenges and Learnings at Scale. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. pp. 1–16. DOI:10.1145/3458817.3476210.

Cepuc A, Botez R, Craciun O, Ivanciu IA and Dobrota V (2020) Implementation of a Continuous Integration and Deployment Pipeline for Containerized Applications in Amazon Web Services Using Jenkins, Ansible and Kubernetes. In: *Proceedings of the 2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. pp. 1–6.

Deelman E, Vahi K, Juve G, Rynge M, Callaghan S, Maechling PJ, Mayani R, Chen W, Ferreira da Silva R, Livny M and Wenger K (2015) Pegasus, a workflow management system for science automation. *Future Generation Computer Systems* 46: 17–35.

Dusia A, Yang Y and Taufer M (2015) Network Quality of Service in Docker Containers. In: *Proceedings of the 2015 IEEE International Conference on Cluster Computing*. pp. 527–528.

en Kiki O, Evans C and Ingerson B (2009) Yaml ain't markup language (yaml) version 1.1. Working Draft 2008.

Factor M, Meth K, Naor D, Rodeh O and Satran J (2005) Object storage: the future building block for storage systems. In: *Proceedings of the 2005 IEEE International Symposium on Mass Storage Systems and Technology*. pp. 119–123.

Group H (2000-2010) Hierarchical data format version 5. http://www.hdfgroup.org/HDF5.

Herbein S, Dusia A, Landwehr A, McDaniel S, Monsalve J, Yang Y, Seelam SR and Taufer M (2016) Resource Management for Running HPC Applications in Container Clouds. In: *High Performance Computing*. Springer International Publishing, pp. 261–278.

Hightower K, Burns B and Beda J (2017) *Kubernetes: Up and Running Dive into the Future of Infrastructure*. O'Reilly Media, Inc.

Jain A, Ong SP, Chen W, Medasani B, Qu X, Kocher M, Brafman M, Petretto G, Rignanese GM, Hautier G, Gunter D and Persson KA (2015) FireWorks: A Dynamic Workflow System Designed for High-Throughput Applications. *Concurr. Comput.: Pract. Exper.* 27(17): 5037–5059.

Keahey K, Anderson J, Zhen Z, Riteau P, Ruth P, Stanzione D, Cevik M, Colleran J, Gunawi HS, Hammock C, Mambretti J, Barnes A, Halbach F, Rocha A and Stubbs J (2020) Lessons Learned from the Chameleon Testbed. In: *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*. USENIX Association.

Lei Q, Guo J, Noureddine A, Wang A, Wuttke S, Brinker CJ and Zhu W (2020) Sol–Gel-Based Advanced Porous Silica Materials for Biomedical Applications. *Advanced Functional Materials* 30(41): 1909539.

Lingerfelt E, Belianinov A, Endeve E, Ovchinnikov O, Somnath S, Borreguero J, Grodowitz N, Park B, Archibald R, Symons C, Kalinin S, Messer O, Shankar M and Jesse S (2016) BEAM: A Computational Workflow System for Managing and Modeling Material Characterization Data in HPC Environments. *Procedia Computer Science* 80: 2276–2280.

Maire E, Buffière JY, Salvo L, Blandin JJ, Ludwig W and Létang JM (2001) On the Application of X-ray Microtomography in the Field of Materials Science. *Advanced Engineering Materials* 3(8): 539–546.

Manvi SS and Krishna Shyam G (2014) Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. *Journal of Network and Computer Applications* 41: 424–440.

McDaniel S, Herbein S and Taufer M (2015) A Two-Tiered Approach to I/O Quality of Service in Docker Containers. In: *Proceedings of the 2015 IEEE International Conference on Cluster Computing*. pp. 490–491.

Merkel D (2014) Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014(239).

Monsalve J, Landwehr A and Taufer M (2015) Dynamic CPU Resource Allocation in Containerized Cloud Environments. In: *2015 IEEE International Conference on Cluster Computing*. pp. 535–536.

Olaya P, Kennedy D, Llamas R, Valera L, Vargas R, Lofstead J and Taufer M (2023) Building Trust in Earth Science Findings through Data Traceability and Results Explainability. *IEEE Transactions on Parallel and Distributed Systems* 34(2): 704–717.

Pascucci V, Scorzelli G, Summa B, Bremer P, Gyulassy A, Christensen C, Philip S and Kumar S (2012) The ViSUS Visualization Framework. In: Bethel EW, Childs H and Hansen CD (eds.) *High Performance Visualization - Enabling Extreme-Scale Scientific Insight*, Chapman and Hall / CRC computational science series. CRC Press.

Penny D (2020) NASA: We forgot the $30m bandwidth charges! https://www.linkedin.com/pulse/oops-we-forgot-30m-bandwidth-charges-david-penny/.

Petruzza S, Venkat A, Morrical N, Scorzelli G and Pascucci V (2020) The OpenVisus Framework for Extreme Data Management, Analysis and Visualization. https://github.com/sci-visus/OpenVisus.

Peukert W, Segets D, Pflug L and Leugering G (2015) *Unified Design Strategies for Particulate Products*, volume 46. pp. 1–81.

Randles BM, Pasquetto IV, Golshan MS and Borgman CL (2017) Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study. In: *Proceedings of the 2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. pp. 1–2.

Rocklin M (2015) Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In: *Proceedings of the 14th Python in Science Conference (SCIPY)*. pp. 126–132.

Sammons G (2016) *Exploring Ansible 2: Fast and Easy Guide*. CreateSpace Independent Publishing Platform. ISBN 9781539751311. URL `https://books.google.com/books?id=vgQiMQAACAAJ`.

Singh R, Graves JA, Anantharaj V and Sukumar SR (2019) Evaluating Scientific Workflow Engines for Data and Compute Intensive Discoveries. In: *Proceedings of the 2019 IEEE*

International Conference on Big Data (Big Data). pp. 4553–4560.

Stodden V, McNutt M, Bailey DH, Deelman E, Gil Y, Hanson B, Heroux MA, Ioannidis JP and Taufer M (2016) Enhancing reproducibility for computational methods. *Science* 354(6317): 1240–1241.

Taghizadeh-Popp M, Kim J, Lemson G, Medvedev D, Raddick M, Szalay A, Thakar A, Booker J, Chhetri C, Dobos L and Rippin M (2020) SciServer: A science platform for astronomy and beyond. *Astronomy and Computing* 33: 100412.

Taufer M, Deelman E, da Silva RF, Estrada T and Hall M (2021a) A Roadmap to Robust Science for High-throughput Applications: The Scientists' Perspective. In: *Proceedings of the 2021 IEEE 17th International Conference on eScience (eScience)*. pp. 247–248.

Taufer M, Deelman E, Silva RFd, Estrada T, Hall M and Livny M (2021b) A Roadmap to Robust Science for High-throughput Applications: The Developers' Perspective. In: *Proceedings of the 2021 IEEE International Conference on Cluster Computing (CLUSTER)*. pp. 807–808.

Wang J, Crawl D, Purawat S, Nguyen M and Altintas I (2015) Big data provenance: Challenges, state of the art and opportunities. In: *Proceedings of the 2015 IEEE International Conference on Big Data (Big Data)*. pp. 2509–2516.

Wilde M, Hategan M, Wozniak JM, Clifford B, Katz DS and Foster I (2011) Swift: A language for distributed parallel scripting. *Parallel Computing* 37(9): 633–652.

Wu F, Wu Q and Tan Y (2015) Workflow Scheduling in Cloud: a Survey. *The Journal of Supercomputing* 71(9): 3373–3418.