End-to-end Integration of Scientific Workflows on Distributed Cyberinfrastructures: Challenges and Lessons Learned with an Earth Science Application

Camila Roa[1], Mats Rynge[2], Paula Olaya[1], Karan Vahi[2], Todd Miller[3], John Goodhue[4], James Griffioen[5], David Hudak[6], Shelley Knuth[7], Alana Romanella[7], Ricardo Llamas[8], Rodrigo Vargas[8], Miron Livny[3], Ewa Deelman[2], Michela Taufer[1]

[1] University of Tennessee, Knoxville, [2] University of Southern California, [3] University of Wisconsin-Madison, [4] Massachusetts Green High Performance Computing Center, [5] University of Kentucky, [6] Ohio Supercomputer Center, [7] University of Colorado, [8] University of Delaware

ABSTRACT

Distributed cyberinfrastructures (CI) pose opportunities and challenges for the execution of scientific workflows, especially in the context of Earth science applications. They provide heterogeneous resources that can meet the needs of the applications that are part of the scientific workflows and provide the necessary performance and scalability to achieve scientific goals. However, the challenge with distributed CI is that it is difficult to find the right resources for the applications and to orchestrate the workflow execution from resource provisioning to job execution to delivering the final results. In some cases, poor choice of resources may result in slow execution or outright failure. In this paper, we present Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) Pegasus, a CI solution built as part of the U.S. National Science Foundation ACCESS program that provides automated execution of scientific applications. We demonstrate Pegasus's capabilities with SOil MOisture SPatial Inference Engine (SOMOSPIE), an earth science multi-component application for fine-grained soil moisture predictions. We identify a roadmap to migrate applications such as SOMOSPIE on ACCESS resources with the support of ACCESS Pegasus, outlining both strengths and weaknesses of this approach.

CCS CONCEPTS

• **Software and its engineering → Distributed systems organizing principles**.

KEYWORDS

Workflows, Machine learning, Soil moisture, High throughput computing, Containers

ACM Reference Format:

Camila Roa[1], Mats Rynge[2], Paula Olaya[1], Karan Vahi[2], Todd Miller[3], John Goodhue[4], James Griffioen[5], David Hudak[6], Shelley Knuth[7], Alana Romanella[7], Ricardo Llamas[8], Rodrigo Vargas[8],, Miron Livny[3], Ewa Deelman[2], Michela Taufer[1] . 2023. End-to-end Integration of Scientific Workflows on Distributed Cyberinfrastructures: Challenges and Lessons Learned with an Earth Science Application. In *2023 IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC '23), December 4–7, 2023, Taormina (Messina), Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3603166.3632142

UCC '23, December 4–7, 2023, Taormina (Messina), Italy

ACM ISBN 979-8-4007-0234-1/23/12…$15.00
https://doi.org/10.1145/3603166.3632142

1 INTRODUCTION

Today, scientists have access to various computing resources to do their work, including high-performance computing (HPC), high-throughput computing (HTC), commercial and academic clouds, and edge and quantum systems. Using these resources, scientists can execute complex, resource-demanding applications to gain insights and knowledge about the world around us and the world we can create. However, the power of these resources is limited to those who know cyberinfrastructure (CI) and how to leverage the CI to execute their applications [3] successfully. The ACCESS initiative, based in the US, seeks to expand the use of CI across a broader and more diverse scientific community. ACCESS is a single entry point for over 20 compute, cloud, storage, and networking systems. While aggregating a large number of resources can benefit research overall, it also makes it more and more difficult for users to choose and combine compute resources effectively. To enable a broad community of scientists with various levels of CI knowledge to leverage the power of distributed CI, we need to provide users with solutions that automate the end-to-end resource provisioning process and workflow management.

This experiential paper presents our solution to this open challenge, ACCESS Pegasus [1], that serves as a central hub for automatic deployment of workflows on ACCESS resources. Specifically, ACCESS Pegasus provides services for *automation* for which workflows automate repetitive and time-consuming tasks, thereby reducing the workload of researchers and avoiding many human errors; *reusability* for which workflows can be used to build libraries of reusable code and tools that other researchers can adapt; *reproducibility* for which workflows allow researchers to document and reproduce their analyses, ensuring their validity; and *scalability* for which workflows allow researchers to scale up their computations to handle large data sets and complex analyses, enabling scientists to tackle more challenging research problems. We pragmatically illustrate ACCESS Pegasus's functionalities by describing the integration process and execution challenges encountered while orchestrating an end-to-end ML-based earth science workflow called

SOMOSPIE [19]. Through a use case studying the soil moisture prediction of an agricultural region such as Oklahoma, we show ACCESS Pegasus's capability to address the need of scientists to execute workflows automatically in heterogeneous environments. This, in turn, empowers users from smaller projects and users with limited research computing experience to join the community and advance scientific discovery through the effective use of available tools and resources. The contributions of this paper are as follows:

- We present ACCESS Pegasus and its integration into the ACCESS software layers and computing resources.
- We demonstrate how earth science applications such as SOMOSPIE can be easily executed on top of ACCESS Pegasus.
- We deploy SOMOSPIE on top of ACCESS Pegasus to predict the soil moisture in a fine-grained resolution for the state of Oklahoma and measure data transfers, short and long-term storage, and memory usage during the execution.
- We extract lessons learned from our use case that apply to a broader range of scientific applications.

In Section 2, we present ACCESS Pegasus with its integration into well-known software packages such as Open OnDemand [11], HTCondor [21] Annex, and the Open Storage Network (OSN) [22] on top of ACCESS resources such as IU JetStream2 [10], Purdue Anvil [20]. In Section 3, we describe how we engineer SOMOSPIE to execute on top of ACCESS Pegasus. In Section 4, we demonstrate the use of SOMOSPIE on top of ACCESS Pegasus for a real case predicting soil moisture for the state of Oklahoma. Last, in Section 5, we explore challenges encountered and lessons learned, and end the paper with conclusions in Section 6.

2 ACCESS PEGASUS SOFTWARE INTEGRATION

We design ACCESS Pegasus as a web-based platform that offers seamless workflow management across ACCESS resources. ACCESS Pegasus interfaces with numerous CI components, such as Open OnDemand and Jupyter for a web interface, CILogon [2] for authentication, Open Storage Network (OSN) for data management [22], HTCondor for workload and resource management [21], and Pegasus [6] for user workflow definition and execution. ACCESS Pegasus deploys user-friendly navigation, featuring comprehensive self-guided Jupyter training modules. Figure 1 shows Pegasus integration with ACCESS: the web browser is used (1) to authenticate (2) and interact with Open OnDemand, Jupyter, and the ACCESS Pegasus APIs. Workflow jobs are handed off to the HTCondor job queue. Users can then provision resources via HTCondor Annex (3') or the IU JetStream Exosphere web interface (3"). Once the pilot jobs are provisioned (4), workflow jobs execute on the resources. The jobs transfer data in/out of the Open Storage Network (5). These modules aid users in creating, submitting, monitoring, and troubleshooting their workflows for effective deployment across ACCESS resources.

2.1 Pegasus Features

ACCESS Pegasus's core is the Pegasus's Workflow Management System (WMS), an engine used to compose and execute multi-step computational workflows. These workflows represent a series of computational tasks or jobs to be executed in a particular order, modeled as a Directed Acyclic Graph (DAG). Pegasus orchestrates end-to-end workflows, managing the scheduling of jobs and the data staging in and out of the components across a local or wide-area network. In contrast to most other workflow systems, Pegasus focuses on data management. This means that Pegasus performs data transfers as part of the workflow, considers data placement when scheduling jobs, and cleans up data when it is no longer required. These data features are crucial to running workflows in distributed environments, such as when ACCESS compute and storage resources are not colocated. Pegasus also serves as a powerful tool for tracking the workflow status and debugging of failed jobs in a workflow. Specifically, we include a variety of monitoring and debugging tools in Pegasus such as Pegasus-monitor, Pegasus-status, and Pegasus-analyzer, to get statistics and essential information related to the execution of end-to-end workflows. This is critical in lowering the barrier of entry to new users and systems. The Pegasus's runtime monitoring daemon, which is launched when the execution of a workflow starts, monitors the end-to-end flow of data, generating the log information needed for the scheduling in an interactive cycle.

2.2 Open OnDemand Web-based Interface

We deploy Open OnDemand as the web-based interface for ACCESS Pegasus to enable users to employ Jupyter Notebooks and command-line interfaces directly from their web browser. Open OnDemand integrates with CILogon, enabling seamless login/authentication for any ACCESS user with a current allocation, thereby simplifying access. This interface allows researchers easy access to high-performance computing (HPC) resources, enabling them to manage their jobs, files, and data directly through a user-friendly web interface, without requiring extensive knowledge of the command-line interface or the intricacies of HPC systems.

2.3 HTCondor Annex Resource Provisioning

ACCESS Pegasus delegates HTCondor [21] Access Point (AP) to execute workflows on ACCESS resources. A cornerstone of ACCESS Pegasus is its ability to bring in resources on demand: to associate an ACCESS allocation with a workflow and make the jobs run under that allocation. In ACCESS Pegasus, we use the new `htcondor annex` command-line tool, which supports a subset of ACCESS resources and uses the collections of capacity method to manage annexes. Management includes creating, monitoring, and shutting down annexes. An annex can shut itself down if it is idle for longer than a configurable amount of time. Users can manually stop using capacity when they no longer have jobs to run, minimizing waste.

2.4 IU Jetstream2 and Purdue Anvil Resources

We deploy two computing resources available on ACCESS for our use cases: IU Jetstream2 and Purdue Anvil. Jetstream2 and Anvil contain features of other available resources (including SDSC Expanse, PSU Bridges2, and TACC Stampede2). Jetstream2 is a cloud resource, whereas Anvil is an HPC (High-Performance Computing) resource. We provide pilot jobs on Jetstream2 via the Exosphere interface1[8]. We initiate virtual machines within their allocations, choosing the suitable size for our scientific use case described in

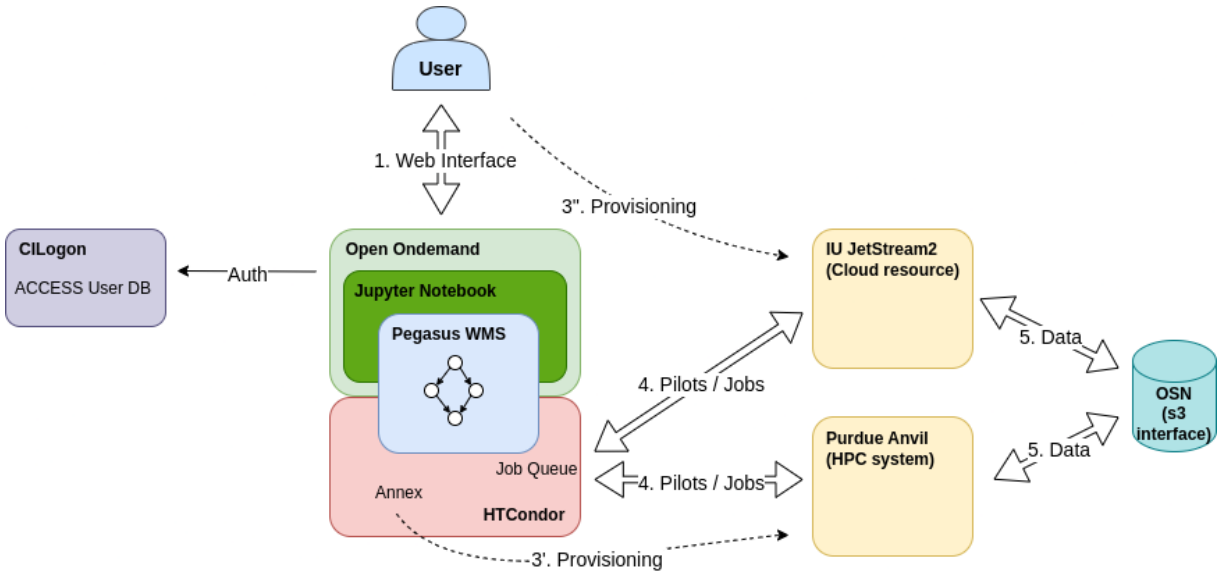

Figure 1: Overview of ACCESS Pegasus integration.

Section 3 in terms of cores, memory, disk space, and so forth. To configure the virtual machine, we only need to supply a token provided by ACCESS Pegasus. The HTCondor Annex enables provisioning on Anvil. We remain within the ACCESS Pegasus environment in this scenario. We leverage the fact that all commands for HTCondor Annex are issued within the Open OnDemand web-based command line.

2.5 Open Storage Network

We leverage the Open Storage Network (OSN)[22] as our storage. OSN functions as a centralized object store, employing an S3-compatible API. For our work, we use OSN for storing intermediate data that are an integral part of the data provenance of our use case execution and for preserving final output data to deliver to the scientists. By segregating the location of intermediate data from computing locations, we can execute our workflows across multiple ACCESS resources. This arrangement allows for the late binding of jobs to available computing resources. In other words, a job can be executed on any ACCESS resource, retrieve data from OSN, perform computations, and store the results back to OSN. Pegasus seamlessly manages the data, including all transactions (gets/puts) with OSN. However, as we will explore in Section 3, this convenient setup does come with a trade-off: a potential decline in performance.

3 SOMOSPIE INTEGRATION

SOMOSPIE is an earth science engine that deploys machine learning (ML) models to predict high-resolution soil moisture from 27km resolution satellite data. SOMOSPIE combines the satellite-based data from the ESA-CCI soil moisture database [17] with hydrologically meaningful terrain parameters of the region of interest to perform the downscaling. The strategy for downscaling based on satellite-derived soil moisture is an alternative to traditional methods that involve simple extrapolation and interpolation based on data from monitoring networks [4, 5, 7]. The advancement of flexible frameworks like SOMOSPIE improves our comprehension of soil moisture changes and their impacts on ecological processes and climate change forecasting on regional and global scales. The finer resolutions provided by SOMOSPIE are required for practical use in applications such as precision forestry and agriculture, hydrology for landscape ecology, and regeneration dynamics, where coarse-resolution data is inadequate [9, 12–14]. SOMOSPIE comprises three components: a terrain parameter generation [18], a data transformation, and an ML-based prediction. Figure 2 shows the full pipeline with the three components.

Building on our previous efforts to enable reproducibility [15] and scalability [16] by leveraging HPC and cloud-converged technology, we integrate SOMOSPIE into ACCESS Pegasus, implementing each component of SOMOSPIE as an ACCESS Pegasus workflow. ACCESS Pegasus enables portability across ACCESS resources. We submit the workflows for execution on a single Jupyter Notebook on two different compute sites. ACCESS Pegasus performs independent resource management after submission by leveraging HTCondor's provisioning capabilities. With Jetstream2, the provisioning is done by starting previously configured VM instances; with Anvil, the HTCondor Annex tool sends pilot jobs to the ACCESS resource providers.

3.1 Terrain Parameter Generation

This SOMOSPIE component predicts the terrain parameters of a region of interest at the desired finer resolution. The component uses a Digital Elevation Model (DEM) of the region of interest downloaded from the USGS 3D Elevation Program [23] as input. The DEM is pre-processed by reprojecting it into a metric-unit coordinate system. Then, it is partitioned into tiles with a buffer region which is introduced to prevent boundary artifacts since computation at a single pixel uses values from adjacent pixels. After

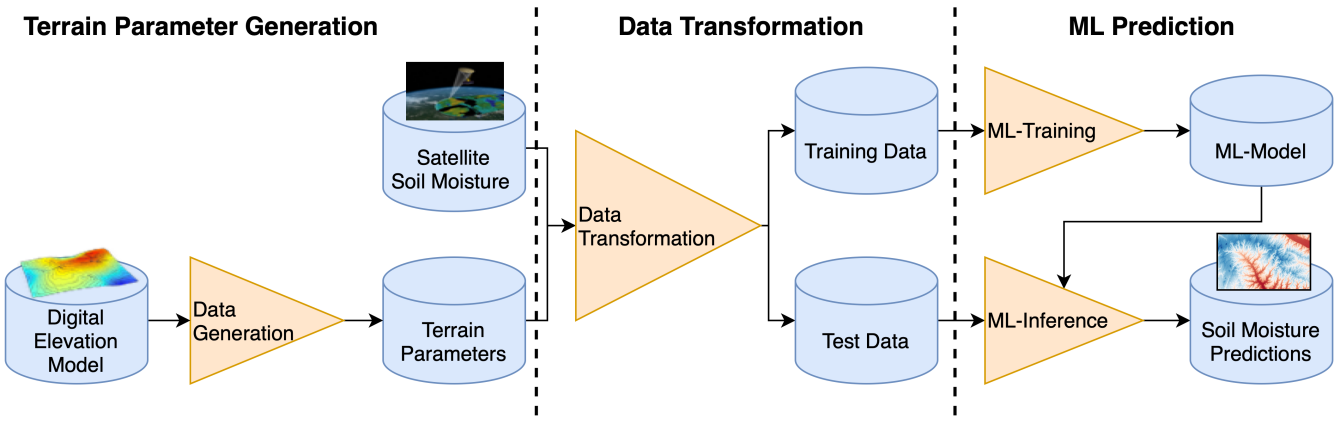

Figure 2: The three SOMOSPIE components.

computing three terrain parameters (i.e., aspect, hillshading, and slope), a mosaic is built using the average values of the overlapping regions within the tiles. Finally, the parameters and the DEM are reprojected to the WGS84 (EPSG:4326) coordinate system to be later combined with the satellite-based soil moisture data.

The ACCESS Pegasus workflow generating terrain parameters is designed using a Jupyter notebook and leverages the Pegasus API. The notebook describes each task, specifying the software used, arguments, necessary input data files, and expected output files. Additionally, we stipulate that tasks run within a Singularity container with the SOMOSPIE software stack installed. Figure 3 shows the resulting Pegasus workflow implementing the terrain parameter generation with its data dependencies and transformations; ellipses represent transformations, green rectangles input data, gray intermediate data, and blue output data.

3.2 Data Transformation

This SOMOSPIE component combines the terrain parameters with satellite-based soil moisture data to generate the input files for the ML-based prediction workflow. The component starts by downloading soil moisture daily values from the ESA-CCI database for a specific year and then computes their monthly averages. Afterward, it projects these averages to match the projection of the terrain parameters (WGS84). Then, it finds the values of the terrain parameters at the points where there is soil moisture data and generates a file with a stack of values for each month. It crops the data to the region of interest. Additionally, it generates a set number of evaluation tiles corresponding to the stacked terrain parameters within the region of interest, which the ML-based prediction uses to generate high-resolution values.

The Pegasus workflow implementing the data transformation follows a similar structure and execution approach as the terrain parameter generation. The choice to separate these into two distinct workflows stemmed from logical divisions in processing (see Figure 2) and the varied frequency at which each component needs to be executed. Figure 4 shows the Pegasus workflow implementing the data transformation with its input, intermediate, and output data.

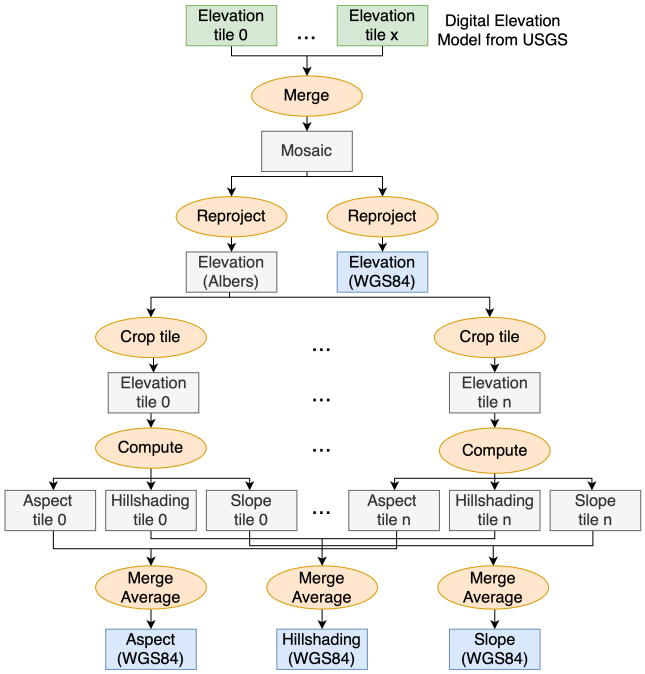

Figure 3: Pegasus workflow implementing the terrain parameter generation component.

3.3 ML-based Prediction

This SOMOSPIE component uses the training and evaluation data from the previous workflow to predict fine-grained soil moisture for the region of interest. We take the training data that combines the satellite soil moisture data and terrain parameters at satellite resolution (27km) and standardize it using the z-score method. We save the fitted scaler and apply it to the evaluation data, which includes the terrain parameters at the desired prediction resolution, so we keep consistency in the standardization. The standardized training data is then used to train and save an ML model (e.g., K-Nearest Neighbor or Random Forest). We apply these models

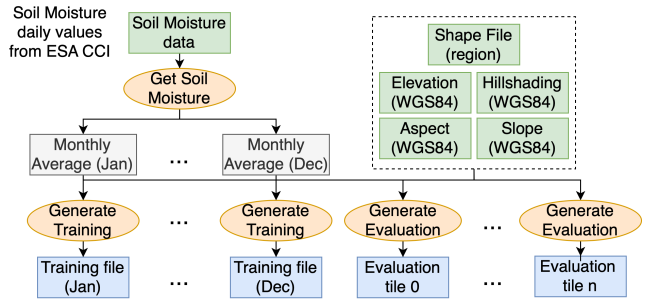

Figure 4: Pegasus workflow implementing the data transformation component.

to the standardized evaluation data to compute high-resolution predictions. As an output of this process, we obtain a mesh of soil moisture values at the aimed resolution for the region of interest.

Like the first two components, the third component is also implemented as a separate Pegasus workflow due to differences in execution frequency. One advantage of this separation is evident when further developing and adding ML models. This division means the scientist can update and execute the ML workflow without re-running the terrain parameter generation and data transformation workflows. While Pegasus allows for data reuse (i.e., re-executing workflows with available partial results), which prunes the DAG based on data availability, collaborators might prefer a clearer logical component division over a more intricate workflow and execution setup. Figure 5 shows the Pegasus workflow implementing the fine-grained predictions with the two available ML models.

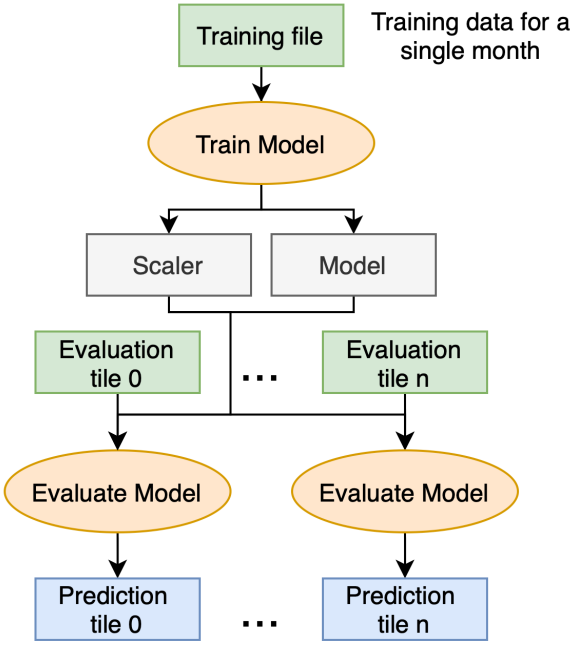

Figure 5: Pegasus workflow implementing the ML-based prediction component.

4 SCIENTIFIC USE CASE

We run SOMOSPIE on top of ACCESS Pegasus to predict Oklahoma's soil moisture values at 10m resolution and measure critical statistics (i.e., workflow wall time, memory, and CPU usage). We select this resolution because it is large enough to raise memory usage concerns when running with the largest CPU instance available in Jetstream2. At the same time, 10m resolution keeps the workflow wall time under 11 hours using one VM and, therefore, does not hinder rapid iteration. Table 1 shows each ACCESS Pegasus workflow's input and output data sizes for the Oklahoma region. For the first workflow, the terrain parameter generation (T.P. Gen.), the input data are the Digital Elevation Models (DEMs) at the resolution we aim to predict, in this case, 10m with a size of 17GB. We use these DEMs to calculate the terrain parameters (i.e., elevation, aspect, hillshading, and slope), generating around 55GB. For the second workflow, the data transformation (Data Tf.), we combine the terrain parameters and satellite soil moisture data to create the training and evaluation data. The training data includes the soil moisture satellite data at 27km and the terrain parameters sampled at the same resolution (27km). Given the coarse resolution in the training files, these files are in the order of KBs. The evaluation data includes the terrain parameters at the resolution for prediction (10m), the largest files in SOMOSPIE (64GB). As we increase the resolution of prediction in a region, more points are covered, translating into larger DEMs, terrain parameters, and evaluation data from GBs to TBs. In contrast, the training data remains the same given that the satellite soil moisture resolution never changes; it is always 27km. We use the training and evaluation data for the third workflow, the ML-based prediction at high resolution (ML Prediction), to get the models and predictions. The models are in the order of MBs, while the predictions are in GBs for our use case.

Table 1: Data scenario for Oklahoma at 10m resolution.

Workflow	Input data	Size	Output data	Size
T.P. Gen.	Digital Elevation Model (39 tiles)	17GB	Elevation	15GB
			Aspect	18GB
			Hillshading	3GB
			Slope	19GB
Data Tf.	Terrain Parameters	55GB	Training data (12 months)	114KB
	Satellite data (1 year)	42MB	Evaluation data (36 tiles)	64GB
ML Prediction	Training data (1 month)	9.5KB	Model and scaler	4MB
	Evaluation data (36 tiles)	64 GB	Predictions (36 tiles)	17GB

We execute the entire pipeline of the three Pegasus workflows (i.e., generating terrain parameters, transforming data, and predicting high-resolution soil moisture values) with Oklahoma at 10m as input data on Jetstream2 and Anvil. We use OSN as the data staging site for both compute resources. The access point is outside the compute site, so it does not share a filesystem with the worker nodes (Jetstream VM instances or Anvil nodes). The jobs run in a local directory inside a container on the worker node, which means the input data gets transferred from OSN, and the output

data gets transferred in the other direction. We use a single VM on Jetstream2 with 64 CPU cores, 250GB RAM, and a volume-backed 2TB disk. As for Anvil, one node with 128 CPU cores, 256GB of RAM, and 480GB disk was requested. The wall time for each of the workflows executed in both sites is shown in Table 2. The workflow wall time is the time that passes between the start of the workflow execution to its end, while the cumulative job wall time is the sum of the wall time of all jobs in the workflow. For the execution of the terrain parameter generation and data transformation components of SOMOSPIE, the difference between the cumulative job wall time and that seen from the submit side reflects the impact of the data transfers and job management overhead on the overall execution. If the workflows were using a local shared file system, then this difference would be potentially smaller, accounting only for the job management overhead since data transfers between sites are eliminated. On the other hand, for the execution of ML-based prediction, the similarity between the wall time and the cumulative wall time represents the non-parallelizable parts of the workflows either because of resource constraints or data dependencies. This lack of parallelism is due to the memory requirements of the Model Evaluation job, as shown in Table 5, which limits the system to run one job at a time because, otherwise, memory capacity is surpassed.

Table 2: Workflow wall time for workflow executions on Jetstream2 and Anvil.

Workflow	Wall time	Jetstream2	Anvil
T.P. Gen.	Workflow	1 h, 8 mins	1 h, 24 mins
	Cumulative job	1 h, 42 mins	1 h, 35 mins
	Cumulative job as seen from submit side	2 h, 42 mins	2 h, 48 mins
Data Tf.	Workflow	57 mins, 0 s	27 mins, 49 s
	Cumulative job	1 h, 3 mins	54 mins, 32 s
	Cumulative job as seen from submit side	23 h, 16 mins	7 h, 9 mins
ML Prediction	Workflow	7 h, 53 mins	3 h, 58 mins
	Cumulative job	7 h, 30 mins	7 h, 16 mins
	Cumulative job as seen from submit side	7 h, 53 mins	7 h, 49 mins

The runtime and memory usage of the jobs from each of the workflows is shown in Tables 3, 4, and 5. We observe that all the transformations use approximately 13GB of memory. Merge Average is the longest job in Figure 3; it involves much computation since it averages the overlapping values from the tiles of the generated terrain parameters and reprojects the resulting mosaic. The workflow executing the data transformation component has the least memory-intensive transformations, allowing for parallelization if provided with sufficient resources. Its longest transformation is Get Soil Moisture because it downloads daily values of soil moisture worldwide from the ESA CCI database, then computes the monthly averages and reprojects them to the same spatial reference system as the terrain parameters computed by the terrain parameter generation. Most of the script uses threading; however, the number of threads that can be instantiated to download the soil moisture data is limited by the FTP server, which restricts the number of requests. In all cases, the runtime on Anvil is shorter, likely due to the larger number of cores per node. The most memory-intensive transformation of SOMOSPIE is part of the workflow performing the ML-based predictions. The Evaluate Model jobs use 132GB of memory at worst on Jetstream2 and 229GB on Anvil. The gap between the mean and the maximum memory use among these jobs comes from the fact that the input tiles are of variable sizes; this disparity can hinder parallelism when the worst case is used as a measure to request memory for every job (see Sec. 5).

Table 3: Resource usage for the workflow generating terrain parameters on Jetstream2 and Anvil.

Resource	Transformation	Runtime (s)		Memory (GB)	
		Mean	Max	Mean	Max
Jetstream2	Merge	171.77	171.77	13.14	13.14
	Reproject	138.88	144.24	13.07	13.06
	Crop	93.10	110.81	12.69	12.69
	Compute	397.06	497.11	13.43	13.88
	Merge Average	787.83	911.66	14.73	14.73
Anvil	Merge	156.84	156.84	13.51	13.51
	Reproject	133.83	138.72	13.51	13.58
	Crop	86.79	104.11	12.99	12.99
	Compute	371.33	471.18	13.89	13.54
	Merge Average	711.94	801.74	15.02	15.02

Table 4: Resource usage for the workflow executing data transformation on Jetstream2 and Anvil.

Resource	Transformation	Runtime (s)		Memory (GB)	
		Mean	Max	Mean	Max
Jetstream2	Get Soil Moisture	646.26	646.26	0.54	0.54
	Generate Training	14.49	16.54	1.09	1.09
	Generate Evaluation	53.88	70.23	3.08	3.22
Anvil	Get Soil Moisture	181.17	181.17	0.54	0.54
	Generate Training	10.04	16.90	1.09	1.09
	Generate Evaluation	49.49	66.27	3.08	3.22

Table 5: Resource usage for the workflow executing ML-based predictions on Jetstream2 and Anvil.

Resource	Transformation	Runtime (s)		Memory (GB)	
		Mean	Max	Mean	Max
Jetstream2	Train Model	6.18	6.18	0.17	0.17
	Evaluate Model	848.77	1153.56	87.58	132.76
Anvil	Train Model	5.58	5.58	0.18	0.18
	Evaluate Model	809.12	1096.36	168.18	229.94

5 CHALLENGES AND LESSONS LEARNED

Most of the complexities of SOMOSPIE come from its data-intensive nature. The data grows due to either the increased resolution at which soil moisture is modeled or the selection of a region of interest with a larger area. The three component workflows handle large amounts of intermediate and output data, which poses challenges in terms of data transfers, short and long-term storage, and memory usage during computation. We discuss the challenges and lessons learned during the integration process and execution of the use case.

5.1 Memory and Storage

5.1.1 Early termination on memory-intensive jobs. There is a persistent challenge related to the early termination of specific jobs when allocating substantial amounts of memory. This issue repeatedly occurs when jobs are terminated without indicating that the underlying problem is related to memory. In these cases, stdout, stderr, and log files offered no insights. Early termination specifically affects two jobs: Merge Average and the evaluation of the ML model. The former relies on the GDAL toolset, while the latter utilizes scikit-learn libraries. Both of these tools have a reputation for being challenging to manage in terms of memory, as there is no simple or direct method to limit memory consumption. We anticipate that potential system configuration adjustments, such as enhancing the *cgroups* configuration within HTCondor, can improve the management of these exceptions. This change is expected to enable the detection and reporting of early termination associated with memory problems.

5.1.2 Memory requirements. Jobs that require large amounts of memory limit the number of tasks that can be run in parallel. For example, the ML-based prediction workflow's job requiring the most memory is the evaluation model. None of the 31 independent jobs from non-empty tiles required for the transformation were performed in parallel when running on Jetstream2; the largest CPU instance with 250GB of memory is not enough to host more than one job at a time given that for each job 130GB were requested to take into account the largest input (tile). The lack of parallelization is evident from the similarity between the workflow wall time and the cumulative job wall time displayed on 2. In contrast, the cumulative job wall time is 1.83 times longer than the workflow wall time for the Anvil run because a single node is larger than the Jetstream VM. Besides performing optimizations at the transformation level to reduce memory usage, such as using lower precision data types or more memory-efficient data structures, the simplest way to enable parallelism is to provision more resources, which can either be a VM with a larger memory capacity, such that multiple jobs can run simultaneously or an increase in the number of VMs in the case of Jetstream2. Despite increasing the number of Jetstream2 VMs from 1 to 2 and 4, the wall time does not decrease by the same amount. This is due to the non-parallelizable parts of the workflows, job management overhead, and data transfers. The workflow executing the terrain parameter generation does not benefit from running on more resources. With one VM, this workflow is already exploiting all the available parallelization.

5.1.3 Storage requirements. Similar to the memory requirements, the storage requirements of a job can prevent other jobs from running on the same machine simultaneously. However, optimizations can be implemented at the transformation level to use less storage, such as using file formats with built-in compression or lowering stored data's precision. SOMOSPIE uses georeferenced data, so GEOTIFF is the preferred file format for storage with LZW compression. At the workflow level, storage requirements can grow due to job interaction. When jobs use the same inputs, data gets staged multiple times, which impacts not only the amount of storage used by the workflow but also the number of data transfers between sites since data is downloaded to the compute site multiple times.

Table 6: Workflow wall time and cost in Jetstream service units as the number of VMs increases.

Workflow	# of VMs	Wall time (h)	Speedup	Cost per VM (SUs)	Cost (SUs)
T.P. Gen.	1	1.21	1.00	39	39
	2	1.26	0.96	40	81
	4	1.18	1.03	38	151
Data Tf.	1	0.95	1.00	30	30
	2	0.76	1.25	24	49
	4	0.43	2.19	14	55
ML Prediction	1	7.88	1.00	252	252
	2	4.50	1.75	144	288
	4	2.55	3.09	82	326

In SOMOSPIE, the workflows implementing the terrain parameter generation and data transformation have this feature.

In the terrain parameter generation, two intermediate files get staged more than one time during the execution of the workflow because multiple jobs use them as input. As shown in Figure 6, the *Mosaic* file is used by 2 Reproject jobs, one that generates one of the workflow outputs and the other that generates elevation in Albers projection which is required to compute the other terrain parameters. For our use case, the *Mosaic* file is 15GB which means 62GB are needed to run both Reproject jobs in parallel in a single machine, 30GB for the input file downloaded twice, 17GB for the *Elevation (Albers)* file and 15GB for the *Elevation (WGS84)* file. The same phenomenon occurs for the Crop Tile jobs since the *Elevation (Albers)* file is used as input by all of them, so the available storage space limits the number of jobs that can run concurrently. Similarly,

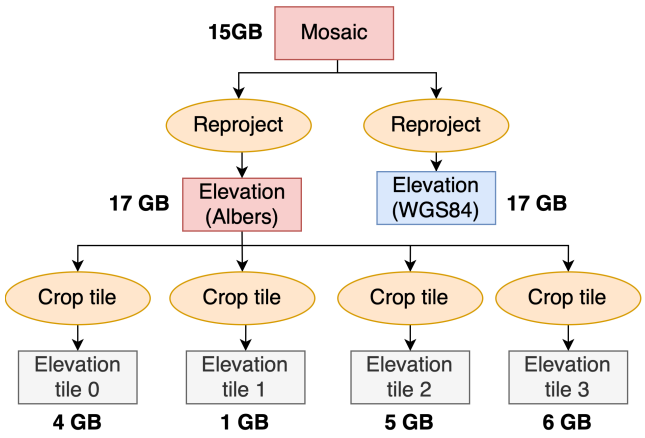

Figure 6: Part of the workflow generating terrain parameters – red rectangles represent files that must be downloaded multiple times to the compute site to run multiple jobs in parallel.

in the data transformation in Figure 7, all of the Generate Train and Generate Evaluation Data jobs use the terrain parameters from the workflow generating terrain parameters as input data (i.e., elevation, aspect, hillshading, and slope) and a file that describes the shape of the region of interest. Neglecting the size of the other inputs in the

order of KB, 55GB of storage is needed only for the input files to run one of the aforementioned jobs.

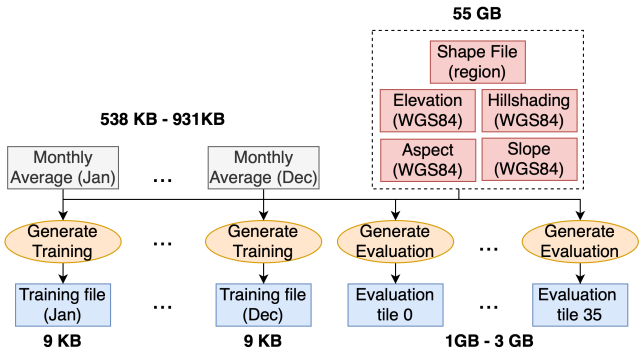

Figure 7: Part of the workflow transforming data – red rectangles represent files that must be downloaded multiple times to the compute site to run multiple jobs in parallel.

One simple solution to enable job-level parallelism is to provision more disk space per individual VM or increase the number of VMs. Another solution is job clustering to ensure data is not downloaded multiple times to the compute site if more than one job uses the same input data, so available disk space is used more efficiently. This enables other jobs, separate from the clustered ones, to run in parallel since disk space is more readily available, reducing workflow wall time.

5.1.4 Customizing requested memory per job. The Evaluate Model transformation in ML-based predictions is notably memory-intensive, with usage ranging from 117GB to a considerable 229GB. This variation in memory consumption between jobs arises due to differences in the input tile sizes. This disparity is attributed to the irregular shape of the region combined with the rectangular tile shape, as depicted in Figure 8. Initially, when constructing the workflow, we provisioned memory for every Evaluate Model job based on the maximum potential need - the largest input. This conservative approach is illustrated in Tables 2 and 5. Allocating large memory blocks that might remain underutilized restricts the number of jobs that can concurrently operate on a single node. Managing resources

Figure 8: Region with irregular shape when cut into tiles.

with unbalanced input sizes can be complex, requiring profiling to grasp the link between input size and transformation memory usage. Even so, this method ensures fewer resources are left idle, fostering job-level parallelism. In our experience with the ML-based prediction workflow, assigning memory per job led to a significant 2-hour and 26-minute cut in total workflow duration, equating to a 1.45x speedup on Jetstream2.

Table 7: Workflow wall time when requesting the same memory for all jobs vs. custom request per job on Jetstream2.

Wall time	Common Request	Custom Request
Workflow	7 h, 53 mins	5 h, 27 mins
Cumulative job	7 h, 30 mins	7 h, 30 mins

5.2 Data Transfer

5.2.1 Compute vs. data transfer time. Optimizations to limit data movement can be of great value for workflows generating terrain parameters and transformation data, which have computations that require large amounts of intermediate data. When scaling up to larger data sizes—in our case, when the increasing resolution or the area of the region of interest—transfers become more costly. Table 8 shows the cumulative job wall time as seen from the submit side and the portion of that metric that represents the time spent on data transfers relative to the wall time spent on computation.

Table 8: Percentage of computing vs. data transfer time.

Resource	Workflow	Wall time (mins)	% Compute	% Transfer
Jetstream2	T.P. Gen.	162	63.0	37.0
	Data Tf.	1396	4.5	95.5
	ML Predict.	473	99.4	0.6
Anvil	T.P. Gen.	168	56.5	43.5
	Data Tf.	429	12.6	87.4
	ML Predict.	469	93.0	7.0

The impact of data transfers is more significant in the workflow transforming data than in the other workflows. Furthermore, the effect is magnified when running on Jetstream2: 95.5% as opposed to 87.4% when running on Anvil. In the case of terrain parameter generation, the data transfers and other overhead account for 37.0% of the cumulative job wall time when running on Jetstream2, while they account for 43.5% on Anvil. As for the ML-based prediction workflow, we can see that the data transfers only play a minor role in the overall wall time being below 7% when execution takes place in either of the two compute sites. The impact of the time that it takes to stage data in OSN for this example workflow makes limiting data movement a priority which can be achieved through optimizations such as clustering, using a shared high performance file system, or bringing compute closer to storage.

5.2.2 Bypass staging. The input of the workflow generating terrain parameters is the Digital Elevation Model (DEM) of the region of interest at a specific resolution, in this case, Oklahoma at 10m. The model comes in the form of multiple files downloaded from the USGS 3D Elevation Program servers. By default, Pegasus downloads this data to the staging site, the OSN object store. After the

input is in the OSN bucket, the Merge job transfers the data to the compute site (i.e., Jetstream2 or Anvil) as part of the job execution. Since the Merge job is the only one in the workflow generating terrain parameters to use this input, one of the data transfers can be avoided without concerns about repeated requests to the USGS servers or longer data staging time for other jobs; we use Pegasus's *"bypass_staging"* option to ensure it directly downloads the input files for this job to the computing site. This decreases 4.12 minutes in the runtime of the transfer jobs related to the Merge job. This decrease is relatively small for this example workflow; however, the effect is magnified when the data scenario of SOMOSPIE changes to a much larger input size. In the case of Oklahoma at 10m, 39 files are downloaded from the USGS servers at 17GB. However, if we run the workflow for the region of the Contiguous United States (CONUS) at the same resolution, the number of files increases to 971 and their size to 480GB. So the savings in the runtime of the transfer jobs related to the Merge job become $1h1min$ when the workflow is executed on Jetstream2. As the problem scales up, the impact of bypass staging becomes more critical.

Table 9: Runtime savings in data transfers for Oklahoma (OK) and the contiguous United States (CONUS) using bypass staging on Jetstream2.

Region	DEM size	Data Transfers Runtime			
		Without bypass	With bypass	Savings	Speedup
OK	17GB	8.88 mins	4.68 mins	4.12 mins	1.90
CONUS	480GB	1h 12mins	11 mins	1h 1mins	6.45

6 CONCLUSIONS AND FUTURE WORK

This paper presents ACCESS Pegasus and its underlying cyberinfrastructure components that enhance accessibility and productivity by integrating with the US-based CI, ACCESS. We demonstrate ACCESS Pegasus's capabilities with SOMOSPIE, an earth science application for high-resolution soil moisture predictions. By deploying ACCESS Pegasus's workflows to execute SOMOSPIE, we identify critical lessons learned when dealing with complex workflows on ACCESS resources, including memory limit management and the automation of transfer optimizations.

Future work builds on the lessons learned and includes moving data closer to the computation process. Workflows such as SOMOSPIE, which deal with terabytes of data, can benefit from minimizing data transfers. A possible solution is clustering tasks in an application pipeline into larger jobs that only require a single transfer of the common data for all tasks. Another solution is to replace the versatile OSN support with CI-specific optimizations (e.g., JetStream2's object store for Jetstream2 runs, and using the Anvil shared work filesystem for Anvil jobs).

ACKNOWLEDGMENTS

The authors acknowledge the support of the National Science Foundation through the awards #1664162, #1841758, #2028923, #2103845, #2103836, #2138811, #2223704, #2330582, #2331152, #2334945, and the ACCESS program through NSF grants #2138286. Results presented in this paper were obtained in part using resources from ACCESS TG-CIS200053 and the IBM Shared University Research Award.

REFERENCES

[1] ACCESS Pegasus Website [n. d.]. ACCESS Pegasus: Automate your Workflow . Available at https://support.access-ci.org/pegasus, [Online; accessed 10-30-2023].
[2] Jim Basney et al. 2019. CILogon: Enabling Federated Identity and Access Management for Scientific Collaborations. *Proceedings of Science (PoS)* 351 (2019).
[3] Alan Blatecky, Damian Clarke, Joel E. Cutcher Gershenfeld, Deborah Dent, Rebecca Hipp, Ana Hunsinger, Al Kuslikas, and Lauren Michael. 2021. *The Missing Millions: Democratizing Computation and Data to Bridge Digital Divides and Increase Access to Science for Underrepresented Communities.* Technical Report. National Science Foundation.
[4] A. Bárdossy and W. Lehmann. 1998. Spatial distribution of soil moisture in a small catchment. Part 1: geostatistical analysis. *Journal of Hydrology* 206, 1 (1998), 1–15.
[5] Hui Chen et al. 2017. Comparison of spatial interpolation methods for soil moisture and its application for monitoring drought. *Environmental Monitoring and Assessment* 189, 10 (Sept. 2017), 525.
[6] Ewa Deelman et al. 2015. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems* 46 (2015), 17–35.
[7] Yuanyuan Ding et al. 2011. Research on the spatial interpolation methods of soil moisture based on GIS. In *Proc. of International Conference on Information Science and Technology*. 709–711.
[8] Exosphere [n. d.]. Exosphere: the User-Friendliest Interface for Non-proprietary Cloud Infrastructure. Available at https://gitlab.com/exosphere/exosphere, [Online; accessed 10-30-2023].
[9] Mario Guevara et al. 2021. Gap-free global annual soil moisture: 15 km grids for 1991–2018. *Earth System Science Data* 13, 4 (2021), 1711–1735.
[10] David Y. Hancock et al. 2021. Jetstream2: Accelerating Cloud Computing via Jetstream. In *Proc. of Practice and Experience in Advanced Research Computing (PEARC '21)*. ACM, Article 11, 8 pages.
[11] Dave Hudak et al. 2018. Open OnDemand: A web-based client portal for HPC centers. *Journal of Open Source Software* 3, 25 (2018), 622.
[12] Ricardo M. Llamas et al. 2020. Spatial Gap-Filling of ESA CCI Satellite-Derived Soil Moisture Based on Geostatistical Techniques and Multiple Regression. *Remote. Sens.* 12, 4 (2020), 665.
[13] Ricardo M. Llamas et al. 2022. Downscaling Satellite Soil Moisture Using a Modular Spatial Inference Framework. *Remote Sensing* 14, 13 (2022).
[14] Ryan McKinney et al. 2015. From HPC performance to climate modeling: Transforming methods for HPC predictions into models of extreme climate conditions. In *Proc. of 2015 IEEE 11th International Conference on e-Science*. IEEE Computer Society, 108–117.
[15] Paula Olaya et al. 2023. Building Trust in Earth Science Findings through Data Traceability and Results Explainability. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 34, 2 (2023), 704–717.
[16] Paula Olaya et al. 2023. Enabling Scalability in the Cloud for Scientific Workflows: An Earth Science Use Case. In *Proc. of 2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE Computer Society, 383–393.
[17] Wolfgang Preimesberger et al. [n. d.]. ESA Soil Moisture Climate Change Initiative: ACTIVE product, Version 08.1. NERC EDS Centre for Environmental Data. Available at https://www.esa-soilmoisture-cci.org, [Online; accessed 02-25-2023].
[18] Camila Roa et al. 2023. 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 (HPDC '23)*. ACM, 311–312.
[19] Danny Rorabaugh et al. 2019. SOMOSPIE: A Modular SOil MOisture SPatial Inference Engine Based on Data-Driven Decisions. In *Proc. of the 15th International Conference on eScience (eScience)*. IEEE, 1–10.
[20] X. Carol Song et al. 2022. Anvil - System Architecture and Experiences from Deployment and Early User Operations. In *Proc. of Practice and Experience in Advanced Research Computing (PEARC '22)*. ACM, Article 23, 9 pages.
[21] Douglas Thain et al. 2005. Distributed computing in practice: the Condor experience. *Concurr. Comput.: Pract. Exper.* 17, 2-4 (2005), 323–356.
[22] The Open Storage Network [n. d.]. OpenStorage Network. Available at https://www.openstoragenetwork.org, [Online; accessed 10-30-2023].
[23] USGS. [n. d.]. 3DEP: 3D Elevation Program. Available at https://apps.nationalmap.gov/downloader/#/elevation, [Online; accessed 10-30-2023].