A Performance Characterization of Scientific Machine Learning Workflows

Patrycja Krawczuk*†¶, George Papadimitriou*†¶, Ryan Tanaka*, Tu Mai Anh Do*† Srujana Subramanya†
Shubham Nagarkar†, Aditi Jain†, Kelsie Lam*, Anirban Mandal‡, Loïc Pottier*, Ewa Deelman*†
*Information Sciences Institute, University of Southern California, Marina Del Rey, CA, USA
{krawczuk, georgpap, tanaka, tudo, lpottier, deelman}@isi.edu
†Computer Science Department, University of Southern California, Los Angeles, CA, USA
{srujanas, slnagark, amjain}@usc.edu
‡Renaissance Computing Institute, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA
anirban@renci.org

2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS) | 978-1-6654-1136-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/WORKS54523.2021.00013

***Abstract*—Scientific workflows are one of the well-established pillars of modern large-scale computational science. More recently, scientists have started to leverage machine learning (ML) capabilities in their workflows, leading to a new category of scientific workflows, denoted as scientific ML workflows. ML is not only about training and inference, modern ML workflows also involve complex data processing steps before the training can start, which are not often accounted for in most performance studies. In this work, we consider scientific ML workflows, from data pre-processing to training, inference, and model evaluation. We aim to explore (i) how scientific ML workflows differ from more traditional scientific workflows and; (ii) how we can characterize ML workflows both in terms of execution time and data movements when executing on a target cloud platform. We select three representative workflows, ranging from image classification to natural language processing and image segmentation, which have been executed using the academic cloud platform, Chameleon. We build four realistic deployment scenarios for each workflow, which stress data movements during workflow executions. Then, we compare the performance observed when utilizing these different configurations and study how different settings impact overall workflows performance and efficiency when running on cloud infrastructures. Finally, we summarize our findings and discuss performance impacts when augmenting scientific workflows with ML techniques and how traditional workflow management systems can improve their support for such workflows.**

***Index Terms*—scientific workflows, machine learning, cloud, data movements, performance characterization**

I. Introduction

Scientific workflows are widely-used abstractions to represent complex computational processes [1]. Scientific workflows (shortened as *workflows* in this work) describe the computational tasks to execute, relative order in which they have to be executed, and how data movements need to be performed. Workflows have played a key role in advancing science by allowing scientists to orchestrate computations at massive scales, from cancer research [2] to molecular dynamics [3]. Traditionally, workflows are modeled as directed acyclic graphs whose vertices represent computational tasks and edges represent data dependencies between tasks [4]. Many workflow management systems (WMS) have been developed ([5], [6]) during the last two decades to express, manage and execute these workflows on distributed infrastructures, high-performance computing (HPC) systems, or clouds. In this paper, we focus on their execution on clouds.

Traditionally, workflows have been used by domain scientists to run large-scale computations. However, the landscape of scientific workflows is rapidly evolving as scientists rely more on machine learning (ML) techniques to model systems and analyze data. This trend, coined as *scientific machine learning* (SciML) [7] by the U.S. Department of Energy has been adopted widely within the scientific community. However, it is the industry that has been driving ML frameworks development (e.g., PyTorch, TensorFlow). Hence, the vast majority of ML pipelines are optimized to run on commercial clouds such as Amazon Web Services or Microsoft Azure.

ML pipelines include not only training and inference steps, but they also require many complex data transformations, i.e. pre-processing steps before the training starts. Most studies do not take into account these extra steps when discussing the performance of an ML application, but these steps matter in terms of complexity and execution time, and will likely grow in significance as ML models become more complex.

Many of the ML pipelines can be viewed as being composed of three main steps: a data processing step, which prepares the data for the second step, the actual training, and then the final step, which traditionally is an evaluation step where the model accuracy and other metrics are verified and potentially visualized. Thus, ML pipelines can easily be expressed as workflows managed by a WMS such as the system we leverage in this paper, Pegasus [5]. In this work, we consider end-to-end ML workflows, from data pre-processing to model evaluation. The research questions at the core of this work are twofold: (*i*) how SciML workflows differ from their more-traditional counterparts? and (*ii*) how, and assuming that SciML workflows are data-intensive, to which extent, data management configurations at a workflow-level impact their performance when running in a cloud environment?

We have chosen three representative ML workflows implemented in Pegasus [5] that solve supervised learning tasks: a

¶Equal contribution

978-1-6654-1136-3/21/$31.00 ©2021 IEEE
DOI 10.1109/WORKS54523.2021.00013

Galaxy images classification workflow, a lung segmentation workflow, and a multimodal (texts and images) social media classification workflow. We have also selected one execution platform, the NSF-funded cloud infrastructure Chameleon [8]. The Chameleon testbed gives us total control over the resources allocated to a workflow and allows us to collect fine-grain monitoring data (execution time, I/O performance, CPU/GPU utilization) about the individual workflow tasks. We use the collected data to characterize each workflow and discuss how these SciML workflows differ from other scientific workflows. To explore the impact of data management solutions on workflow performance, we explore several data management scenarios (e.g., network file systems, non-shared file systems). This paper makes the following contributions:

- A general characterization study of ML workflows based on statistics collected from the three realistic workflows executed using Pegasus WMS;
- An experimental study demonstrating how data management configurations affect ML workflow performance when executing in cloud environments;
- A discussion of the challenges WMSs will be facing when supporting ML workflows and how they could tackle these research questions;
- Dissemination of the set of workflows used in this study as open-source code available online to enable sharing and reproducibility.

II. Workflows

In this section, we introduce each of the workflows and discuss their general characteristics. The workflows feature different scientific domains (astronomy, medicine, and crisis computing) and different supervised learning approaches (image classification, image segmentation, and natural language processing). We decouple data pre-processing and training tasks as data transformations for scientific experiments often require specialized software. The pre-processing tasks in the workflows exhibit data parallelism and can be executed simultaneously across available compute resources. The hyper-parameters optimization (HPO) trials (where different values of hyper-parameters are evaluated) could also be executed in parallel, however, for this study, we use Bayesian optimization and execute the trials sequentially. The inference is treated as a separate task so that the job can be easily deployed when a trained model is used for predictions. The presented workflows are available online with detailed instructions; interested users can replicate our findings.

A. Galaxy Classification Workflow

Context. Scientists use astronomical data from large-scale surveys such as the Sloan Digital Sky Survey (SDSS) to, among other tasks, map astronomical objects and study their evolution. The galaxy morphology classification is a critical step towards understanding how galaxies form and evolve. Due to massive amounts of data currently available, researchers aim to develop robust deep learning models to automatically classify images of the galaxies into 1 of 5 categories: completely round-smooth, in-between smooth, cigar-shaped smooth, edge-on, and spiral. The implementation of this workflow is based on a publication [9] and is publicly available [10].

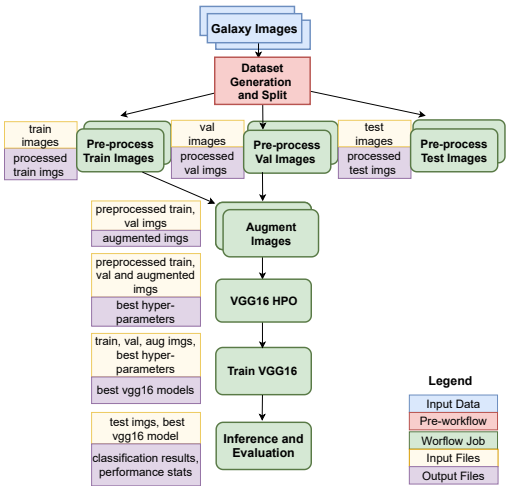

Figure 1. Galaxy Classification Workflow [10].

Workflow Overview. The Galaxy Workflow (Figure 1) utilizes the Galaxy Zoo 2 dataset that consists of 61,578 RGB images, each of size 424x424x3 pixels (1.9 GB of compressed data). The first stage of the workflow (*Dataset Generation and Split*) filters out galaxies based on their feature scores. This reduced dataset of 28,790 images is split into training, validation, and test sets. These datasets are passed to *Pre-process Images* jobs where several data transformations (e.g., crop, downscale, whitening) are applied. To address the problem of class imbalance in the dataset *Augment Images* jobs generate additional instances of underrepresented galaxy types. Next, *VGG16 HPO* job utilizes the Optuna [11], an HPO framework, to find a good set of hyperparameters (e.g., learning rate, numbers of transferred layers). The chosen hyperparameters and all the data are sent to the *Train VGG16* job where the model is trained with the chosen hyper-parameters. The weights of the trained model are saved to a checkpoint file. Finally, the *Inference and Evaluation* job runs predictions on the test set, generates statistics and plots that provide insights into the quality of the trained model.

B. Lung Segmentation Workflow

Context. Precise detection of the borders of organs and lesions in medical images such as X-rays, CT, or MRI scans is an essential step towards correct diagnosis and treatment planning. We implement a workflow that employs supervised learning techniques to locate lungs on X-ray images. The implementation is publicly available [12].

Workflow Overview. The Lung Segmentation Workflow (Figure 2) uses a the *Chest X-ray Masks and Labels* dataset (800 high-resolution X-ray images and masks, 5.4 GB) available on Kaggle. The dataset is split into training, validation, and test sets before the workflow starts. Each set consists of

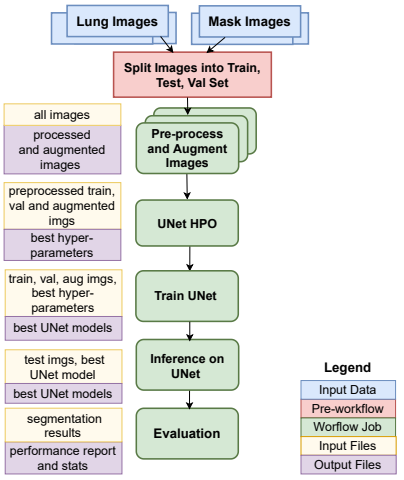

Figure 2. Lung Segmentation Workflow [12].

original lung images (3000x2933 pixels each, 6.3 MB in size) and their associated masks (same resolution, 30 KB in size). The *Pre-process and Augment Images* job resizes images (lungs and masks) to 256x256 pixels and normalizes lung X-rays. Additionally, for each pair of lung image and mask in the train dataset, two new pairs are generated through image augmentation (e.g., rotations, flips). Next, the train and validation data are passed to the *UNet HPO* job, where Optuna [11] explores different learning rates. The *Train UNet* job fine-tunes the UNet model with the recommended learning rate on the concatenated train and validation set, and saves the weights into a file. The *Inference on Unet* job uses the trained model to generate masks for the test X-ray images. The final step of the workflow, the *Evaluation* job generates a PDF file with the scores for relevant performance metrics and prints examples of lung segmentation images produced by the model. As the inference and evaluation steps are implemented as separated jobs, the *Inference on Unet* job can be deployed independently for real-world predictions.

C. Crisis Computing Workflow

Context. In recent years, social media (SM) platforms like Twitter and Instagram have proven to be valuable sources of critical information during disaster events. The published multi-modal content can provide timely and actionable information to local officials. The growing field of crisis informatics focuses on developing deep learning methods to automate extraction of valuable posts from SM threads. Our implementation is inspired by [14] and available publicly [13].

Workflow Overview. The workflow consists of the two pipelines that ingest, respectively, pictorial and textual parts of SM posts. We use the CrisisMMD v2.0 [15] datasets (18,082 images and 16,058 texts, about 2 GB of data) and its accompanying data-split files. The textual part of the tweets is passed to the *Pre-process Tweet Text* job, where stop words, special symbols, and links are removed. Then, the train set of clean tweets are embedded using GloVe [16] 200-d Twitter

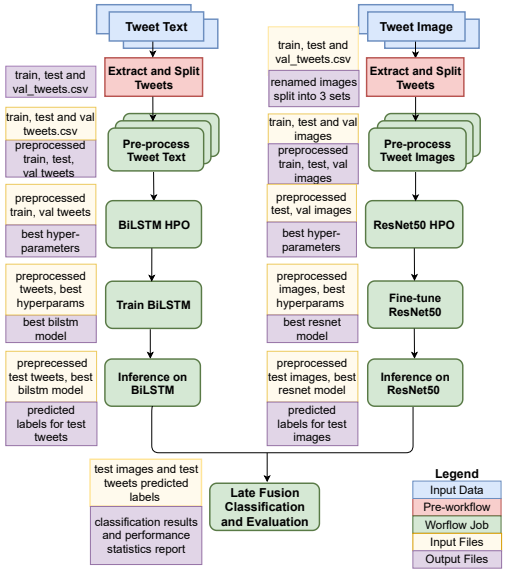

Figure 3. Crisis Computing Workflow [13].

pre-trained vectors (758 MB) and then used to find the best hyper-parameters to train the BiLSTM architecture (*BiLSTM HPO* job). Next, the model is re-trained on the concatenated train and validation datasets, and its weights are saved and passed to the *Inference on BiLSTM* job. Here, the model predicts labels for the text tweets. A similar procedure is used for the image pipeline where, instead of BiLSTM, a ResNet50 model is used. Then, results from both inference jobs are used in the *Late Fusion Classification and Evaluation* job to make final predictions where information is classified as either *informative* or *non-informative*.

Table I presents an overview of models and selected ML hyperparameters used for each of the workflows.

Table I
MACHINE LEARNING HYPERPARAMETERS

Model	Size	#Params	#Trials	#Epochs	Batch Size
Galaxy Classification Workflow					
VGG-16	528 MB	138M	2	5	32
Lung Segmentation Workflow					
UNet	81.6 MB	24.4M	10	25	32
Crisis Computing Workflow					
ResNet50	98 MB	25M	2	5	8
BiLSTM	9 MB	1M	2	10	128

III. EXPERIMENTAL SETUP

In this section, we introduce the Pegasus WMS, which we use to execute all the workflows on our target execution platform, Chameleon. We also describe different data management scenarios that will be used to discover potential bottlenecks that arise when running complex machine learning workflows on a distributed infrastructure.

A. Pegasus Workflow Management System

Pegasus [5] is a popular workflow management system that enables users to design workflows at a high-level of

abstraction. The workflow descriptions developed by the users are independent of the resources available to execute the workflow tasks and are also independent of the location of data and executables. Pegasus relies on HTCondor [17], a job management system particularly well suited for distributed high throughout computing (HTC) environments, to run and manage the generated workflows. Pegasus supports online monitoring via its Panorama branch [18]. Panorama enables end-to-end online workflow monitoring and provides execution traces of the computational tasks (CPU and GPU), statistics for individual transfers and infrastructure-related metrics.

B. Chameleon Cloud Platform

The NSF Chameleon Cloud [8] is a large-scale, deeply programmable testbed designed for systems and networking experiments. Chameleon leverages OpenStack to deploy isolated instances of cloud resources for user experiments. It provides large amounts of heterogeneous compute, storage, and networking resources spread mainly across two sites: University of Chicago (UC) and the Texas Advanced Computing Center (TACC). Across these two sites Chameleon offers over 15K cores and 5 PB storage and users have the ability to provision bare metal compute nodes with custom system configurations connected to user-controlled OpenFlow switches operating at up to 100 Gbps.

We provision and configure selected Chameleon resources to have two environments: (*i*) an environment without a shared filesystem and (*ii*)) an environment with a shared filesystem. The non-shared filesystem setup (see Figure 4) consists of one node located in Texas Advanced Computing Center (TACC) acting as a workflow submit node and HTTP server, and three worker nodes located in University of Chicago (UChicago). The submit node hosts the WMS, which coordinates and launches workflow jobs on the worker nodes. All of the nodes are bare metal nodes with 24 physical cores (hyperthreading disabled), 192GB of RAM, 10Gbps network connection and two of the worker nodes are equipped with one NVIDIA RTX6000 (24GB memory) each. The shared filesystem setup

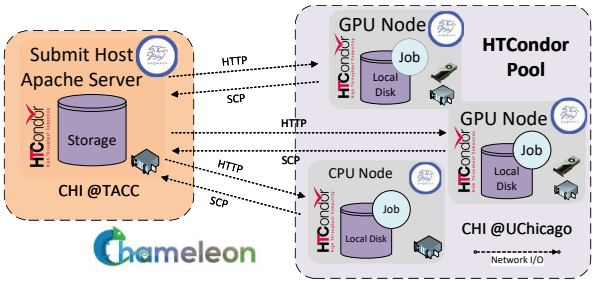

Figure 4. Non-shared filesystem deployment on Chameleon.

(see Figure 5) is deployed entirely within Chameleon at UChicago and uses the same amount of compute resources as the non shared filesystem setup. This time, however, the submit node is configured with a Network Filesystem (NFS) to enable data access from the workers and also serve as the temporary compute scratch location. Finally, both deployments use the same operating system (Ubuntu 18.04) and software stack (HTCondor v8.8.9 and Pegasus Panorama-branch v5.1.0 [18]).

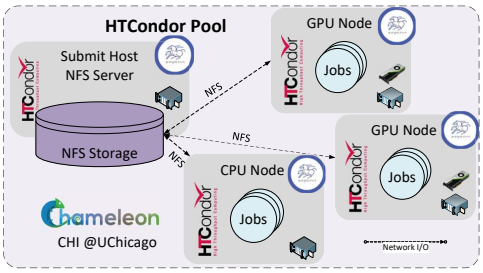

Figure 5. Shared filesystem deployment on Chameleon.

All experiments run the workflows using Pegasus [19] and Singularity containers [20]. We collect statistics using Pegasus' Panorama online architecture [21], execute each workflows 10 times per configuration, and present average and standard deviation results over these 10 runs where applicable. During all the workflow runs there were always enough resources to handle all the jobs added to the queue.

C. Execution Scenarios

We design four experiments that exhibit different data placement and file access strategies.

Baseline. This first scenario, which acts as a baseline scenario, involves executing the workflows without any data placement optimization. Input and intermediate data staging are transferred via *http* from the submit host, while output files are sent back to submit host's staging area via *scp* (see Figure 4). To conduct the transfers we are using a maximum of 8 number of threads (Table II) and we configure the preprocessing tasks to be split in up to 6 jobs.

Container Installed. This second scenario is a variation of the **Baseline** scenario optimized for container image placement. Here, the Singularity container images are pre-loaded on the worker nodes and the jobs are able to pick them up from local disk whereas in the **Baseline** scenario container images are sent over the network before each job.

Clustering. This third scenario attempts to minimize transfers from and to the staging area and optimize data reuse among tasks, while using the transfer mechanisms of the baseline (see Figure 4). To achieve this, we leverage clustering techniques offered by Pegasus and cluster together tasks that are either using the same inputs or intermediate files. Specifically, for the Lung Segmentation and Galaxy Classification workflows we cluster all tasks in a single job, however, for the Crisis Computing workflow we create two clustered jobs, one for the image pipeline and one for the text pipeline, and we leave the late fusion as a single job. In the Clustering scenario we assign an entire node to each cluster of tasks, and we used a maximum of 24 threads to conduct transfers (Table II)

Network Filesystem (NFS). Finally, the fourth scenario uses a shared filesystem among all the nodes, hosted on the submit node (Figure 5). Here, we use a shared file system to host all the input, intermediate and output data, but we also use it as the scratch location during execution. No clustering is employed and the input files are symlinked to the scratch location of the jobs. With this scenario, we attempt to highlight the penalties ML pipelines face when using a shared file system.

Table II
EXECUTABLE WORKFLOW SCENARIOS AND TRANSFERS SETTINGS

Workflow	Scenario	Jobs	Aux. Jobs[1]	Transfer Threads[2]	Files Staged In
Galaxy Classification	Baseline	11	12	8	111950
	Container Inst.	11	12	8	111950
	NFS	11	12	8	111938
	Clustering	1	4	24	28803
Lung Segmentation	Baseline	7	12	8	9446
	Container Inst.	7	12	8	9446
	NFS	7	12	8	9438
	Clustering	1	4	24	1417
Crisis Computing	Baseline	16	12	8	51006
	Container Inst.	16	12	8	51006
	NFS	16	12	8	50987
	Clustering	3	6	24	12758

[1] The number of auxiliary jobs generated by Pegasus during planning.
[2] Number of threads used by Pegasus to transfer (stage in/out) job input files.

IV. GENERAL CHARACTERIZATION

We first aim to characterize each workflow at a global level. To that end, each workflow is executed only with **Baseline** settings. The characterization data is collected at job- and workflow-level. We use Panorama to capture workflow's I/O behavior, CPU utilization and GPU utilization traces. The values presented in Tables III and IV are averaged over 10 runs and I/O data collected.

A. Job-level Characterization

First, we examine the *Pre-processing* job. The data transformations performed during pre-processing are commonly executed on the CPU. This step is easily parallelizable, and data (stored in files) are read into the memory in batches resulting in low values of peak memory. The Lung Segmentation Workflow has the highest peak memory and average execution time as medical images segmentation requires high-resolution images. The pre-processing job in the Crisis Computing Workflow has the highest average CPU utilization due to use of a computationally expensive interpolation algorithm.

The peak GPU memory remains constant for each workflow across the *HPO*, *Training* and *Inference* jobs. That value is bound by the number of the parameters in the model, batch size (see Table I), and size of a data instance. It differs for the *Inference* job on ResNet50 as predictions were calculated on the CPU. The *HPO* and *Training* jobs are the most computationally expensive. The tasks require forward pass of the data and weights update through backward propagation (both consists of many tensor operations). The *Inference* jobs are characterized by lower average GPU utilization as only the forward pass of the data is needed to make predictions. The *HPO* and *Training* jobs for the BiLSTM model expect a large file with the pre-trained sentence-level embeddings as input (a common practice for NLP tasks). Once the model is trained, the required fine-tuned embeddings are stored within the model's weights resulting in a low I/O read value for the BiLSTM *Inference* task.

The *Evaluation* jobs generate performance metrics and plots that help assess the quality and robustness of the trained models. Inference and evaluation steps are sometimes performed in the same script during model development.However, these tasks are often separated when a model is deployed in production. The *Evaluation* jobs are not computationally expensive but with the growing stress on trust and explainability in AI, we expect the complexity of the evaluation to increase in the future and as a result, the cost will grow as well.

B. Workflow-level Characterization

The Crisis Computing Workflow is characterized by the largest amount of both the CPU and GPU hours. This can be attributed to the structure of the workflow that consists of two training pipelines. Moreover, in the image pipeline, we employ "on the fly" image augmentation (i.e., new versions of images are generated on CPU during training and are transferred to GPU), which further increases the number of CPU hours.

The images used in the Galaxy Classification workflow have a low resolution resulting in the small size of the workflow's input. The galaxies are classified based on their shape as presented in the image, and this tasks is not as intricate as deciding whether an image is informative. It also does not require high quality images for complex feature extraction.

As seen in Table IV, the workflows are using many input files, ranging from a couple thousand (Lung Segmentation) to over 28,000 (Galaxy Classification), that are sometimes small in size. Thus, different data management strategies with a workflow, when data moves between tasks, might affect the overall workflow execution. In the next section, we explore 4 different scenarios that are supported by Pegasus WMS and optimize for data access and data placement.

V. EXPERIMENTS

Now that we have defined and characterized our different workflows using a basic configuration, we study the impact of data management on end-to-end workflow performance using different scenarios defined in Section III-C.

Results. For each workflow and each execution scenario, we analyze the total time the workflows took to complete, the cumulative compute time spent on the jobs and the cumulative time spent in staging in/out data for the computations.

In Figure 6, the cumulative compute time remains fairly similar across all workflows for all scenarios, except NFS. In the NFS configuration apart from input data being picked up from the shared location, the NFS was used as the execution's scratch location and all outputs were produced directly to the shared filesystem. This slowed down the compute time, since some of the jobs (e.g, HPO, Train) were I/O heavy (Table III).

Table III
JOB-LEVEL CHARACTERIZATION WHEN RUNNING WITH NON SHARED FILESYSTEM (BASELINE SCENARIO)

Job	I/O Read (MB)	I/O Write (MB)	Avg. CPU (%)[1]	Peak Memory (GB)	Avg. GPU (%)	Peak GPU Memory (GB)[2]	Avg. Exec. Time (Sec)[3]
Galaxy Classification Workflow							
Preprocessing	435.47	260.71	99.85	0.08	-	-	135.21
HPO	2900.37	3404.28	1426.22	41.19	53.22	4.13	3421.48
Training	1754.6	2091.05	789.51	18.8	68.92	4.13	1453.53
Inference & Evaluation	1440.85	527.82	404.01	3.78	26.76	4.42	51.84
Lung Segmentation Workflow							
Preprocessing	8107.44	143.99	120	0.37	-	-	337.85
HPO	3816.46	84.8	100.75	5.96	68.19	22.99	4947.24
Training	540.71	8010.01	104.83	5.12	62.48	22.99	557.4
Inference	396.57	0.86	106.89	3.1	10.39	22.99	22.51
Evaluation	0.39	0.37	129.44	0.43	-	-	6.13
Crisis Computing Workflow							
Preprocessing (image)	1185.06	1457.79	660.74	0.07	-	-	212.48
Preprocessing (text)	11.84	1.12	121.99	0.133	-	-	8.8
HPO (ResNet50)	28321.72	786.75	186.22	6.44	68.73	1.67	1424.91
HPO (BiLSTM)	4464.78	0.95	269.51	4.13	20.92	22.64	1031.6
Training (ResNet50)	17702.71	728.70	179.75	5.94	63.73	1.76	871.99
Training (BiLSTM)	2032.84	5.84	272.56	4.03	20.9	22.64	625.09
Inference (ResNet50)	2734.55	231.07	2386.63	16.64	-	-	3003.4
Inference (BiLSTM)	400.48	0.58	111.57	4.13	42.41	22.64	65.95
Evaluation	384.47	58.59	161.01	11.18	3.72	0.83	211.76

[1] Average CPU utilization of a job type (e.g. HPO, Training), calculated as $100 \times (avg.stime + avg.utime)/(avg.execution\_time)$. If multiple jobs are under the same type, we average their respective CPU utilization.
[2] Maximum video memory used by this type of job across all the workflow runs (here 10).
[3] Average execution time of a job type, it considers only time spent on compute. If multiple jobs fall under this job type we calculate this value as the total sum of the jobs average execution time, as if they were executed sequentially.

Table IV
WORKFLOW EXECUTION PROFILES. (BASELINE SCENARIO)

Workflow	Jobs	Aux. Jobs	Input Files	Input Size (GB)	Container Size (GB)	I/O Read (GB)	I/O Write (GB)	Peak Memory (GB)[1]	Peak GPU Memory (GB)	CPU Hours[2]	GPU Hours
Galaxy Classification	11	12	28793	0.374	2.4	6.29	6.14	41.19	4.42	33.23	1.93
Lung Segmentation	7	12	1408	3.6	4.1	12.56	8.05	5.96	22.99	37.63	1.64
Crisis Computing	16	12	12747	3.2	4.6	55.89	3.2	16.74	22.64	48.76	2.45

[1] Maximum resident size memory used by any job of the workflow.
[2] Total time a single CPU core was assigned to the workflow (for both compute and auxiliary tasks). Same for GPU hours with a single GPU.

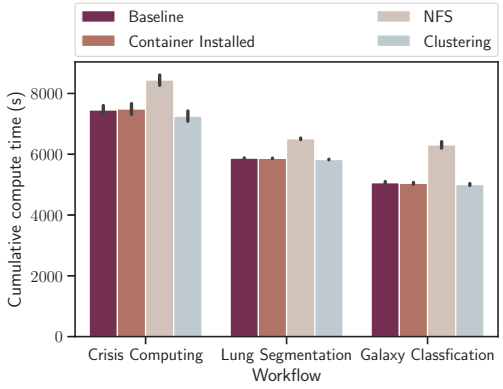

Figure 6. Cumulative compute time for each workflow when running on Chameleon Cloud.

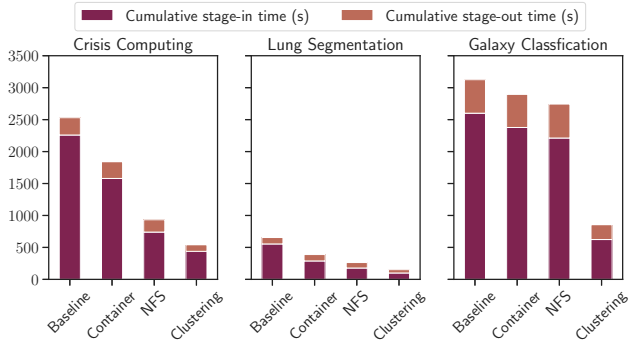

Figure 7. Cumulative stage-in and stage-out time for each workflow when running on Chameleon Cloud.

The results in Figure 7 were anticipated, since each scenario was designed to further optimize time spent on staging in and out data. For all the workflows a reduction in staging time is observed moving from the Baseline towards the Clustering scenario, with these two having the slowest and the fastest times. Even though all of the workflows show an overall improvement across the scenarios, it is not in the same proportion. For the Crisis Computing workflow the *Baseline* was improved by 700 seconds when we moved to the *Container Installed* scenario and a further 800 seconds when we moved to the *NFS* scenario. On the other hand, the Galaxy Classification workflow was improved by only a few hundred seconds. This result can be attributed to the difference in container size used for each of the workflows

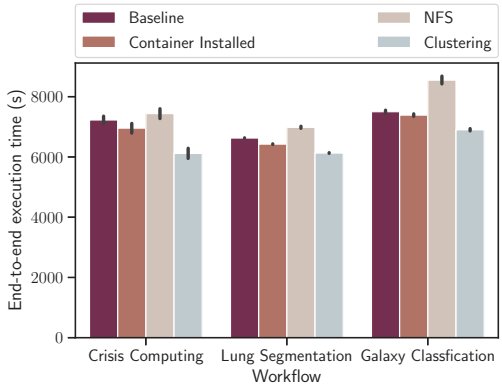

Figure 8. Workflows end-to-end execution time for each scenario when running on Chameleon Cloud.

(4.6 GB vs 2.4 GB), as well as the difference in number of input files (12,747 vs 28,793) and total input size (3.2 GB vs 0.374 GB) - the Galaxy Classification workflow has many small files. Additionally the NFS case is not the fastest of them all, since Pegasus had to transfer files for every compute job, while for the Clustering case Pegasus had to transfer files only for each cluster (Table II). Finally, in Figure 8 the data placement and stage in/out optimizations affect all the workflows and a reduction in end-to-end execution time is observed from Baseline to Container to Clustering. Although, the NFS case had faster stage in/out times than the Baseline and Container Installed scenarios, the overhead in compute time overshadows the staging gains and results in the slowest end-to-end execution time.

Summary. Based on the results, we conclude that clustering together tasks that use the same files should be a first class feature of WMSs that aim to support large, scientific ML workflows. The ease of use and convenience of shared NFS filesystem can be out-weighted by accessing files over the network and sharing the I/O bandwidth among concurrent jobs.

ML workflows rely on multiple library dependencies, and thus support for containers is very important in order to obtain a common execution environment across execution sites and workers. However, the size of these containers can be significant to the size of input and they should not be overlooked. In our experiments preloading the containers didn't improve the end-to-end execution time by a considerable margin, but for workflows like the Galaxy Computing it can make a difference on the network utilization. Finally, many traditional WMS (including Pegasus) anticipate that all the input and output files for each job are known before the workflow execution starts. However, because ML methods often rely on thousands of data files accumulated from different sources the quality of each data file cannot be guaranteed or checked beforehand. To support ML workflows, scientific WMS will have to embrace dynamic changes on the number of input and output files.

VI. Related Work

Scientific workflows and workflow management systems have been two extremely fertile research domains in the last decades and resulted in many contributions (a relevant survey [22]). However, recently, several workflow management systems have emerged to accommodate the life-cycle of ML pipelines on high-performance computing environments and commercial clouds. Wozniak et al. [2] designed a workflow framework for cancer research optimized for HPC resources. Many cloud-native WMS have also been introduced. MLFlow [23] and Pachyderm [24] are few examples of cloud-native Kubernetes [25]-based solutions tailored for ML workflows. Despite this, major companies are also rolling out there own cloud native WMS in support of ML, with some examples being Uber (Michelangelo) [26] and Facebook (FBLearner) [27]. These solutions target pure ML pipelines, where all the pre-processing is done *on-the-fly*, traditionally in Python, and not scientific ML workflows which may contain additional processing steps relying on specialized software.

Apart from WMS, general purpose distributed computing frameworks such as Spark [28] and Hadoop [29] have also been thoroughly investigated for ML workloads [30]. More recently research focus has been shifted towards specialized ML frameworks built on top of serverless computing [31], [32]. Carreira et al. [31] have proposed Cirrus, a framework that facilitates the execution of ML workflows on the cloud using a serverless approach.

Monitoring and profiling of ML pipelines have also been topics with a lot of activity. Zhou et al [33] proposed an extension of HPCToolkit to enable fine-grained performance analysis on GPUs by collecting program counter samples on both CPU and GPU. Profiling and visualization tools have also been introduced by hardware vendors (e.g., NVIDIA with Nsight [34] its in-house profiling tool for GPU application) and ML frameworks such as PyTorch [35] which, when coupled to TensorBoard [36], can help scientists analyze and understand ML algorithm behaviors and performance. However, these tools are focusing on in-depth performance analysis and debugging, in this paper, we used a more practical resource monitoring approach during the ML workflow executions and did not delve into fine-grained analysis of function calls.

A vast majority of existing characterization and performance studies focus exclusively on the model training or inference tasks, and use reference implementations of popular deep learning models that fail to mimic the complexity of workloads used in scientific experiments [37], [38], [39]. To the best of our knowledge, very few solutions have been developed with an academic mindset. This work is one of the first to explore the feasibility of managing scientific ML workflows using WMS primarily designed to support scientific workflows.

VII. Conclusion

We have characterized three representative scientific machine learning workflows from different scientific domains. We used the experimental cloud testbed Chameleon to explore the impact of several data management configurations on the workflows performance. We discussed these results in terms of workflow characterization and in terms of broader implications

on workflow management systems targeting the support of scientific machine learning workflow.

As expected, regardless of their relative small input size, scientific ML workflows can be quite data intensive. The Galaxy workflow, for example, reads and writes more than 13 times it's input size. On the other hand, even though they get a great speed up from GPUs they do not utilize them to their maximum capacity. Furthermore, GPU utilization greatly varies from one ML workflow to another (from 21% to 68%). We have demonstrated that data placement, especially having data collocated (achieved with job clustering), can lead to better execution time, up to 16% compared to the baseline for the Crisis Computing workflow. Finally, we have showed that the convenience of a network shared filesystem incurs a significant performance penalty (up to 13% compared to the baseline for the Galaxy workflow). Although the workflows are represented in Pegasus, the results are not specific to it.

A short-term direction is to investigate more data management configurations with Pegasus (e.g, object-storage) utilizing machines with different I/O systems to assess the confirm our findings that task co-location achieved with Pegasus clustering can significantly improve workflows execution time compared to naive deployments. A longer-term future work direction is to integrate our findings into Pegasus WMS to enhance its support of scientific ML workflows, for example, by providing automatic clustering capabilities that will maximize data locality. The presented workflows are available online with detailed instructions; interested users can replicate our findings.

Acknowledgments. This work is funded by DOE contracts #DE-SC0012636, #DE-SC0022328 and NSF contract #1664162. Results presented in this paper were obtained using the Chameleon testbed supported by NSF.

REFERENCES

[1] R. F. da Silva *et al.*, "Workflows community summit: Bringing the scientific workflows community together," *arXiv:2103.09181*, 2021.

[2] J. M. Wozniak *et al.*, "Candle/supervisor: A workflow framework for machine learning applied to cancer research," *BMC bioinformatics*, vol. 19, no. 18, 2018.

[3] G. Sivaraman *et al.*, "A machine learning workflow for molecular analysis: application to melting points," *Machine Learning: Science and Technology*, vol. 1, no. 2, p. 025015, 2020.

[4] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *ACM Sigmod Record*, vol. 34, no. 3, pp. 44–49, 2005.

[5] E. Deelman *et al.*, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, 2015, funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575.

[6] P. Di Tommaso *et al.*, "Nextflow enables reproducible computational workflows," *Nature biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[7] N. Baker *et al.*, "Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence," US Department of Energy, Tech. Rep., 2 2019.

[8] K. Keahey *et al.*, *Chameleon: A Scalable Production Testbed for Computer Science Research*. CRC Press, 05 2019, pp. 123–148.

[9] X.-P. Zhu *et al.*, "Galaxy morphology classification with deep convolutional neural networks," *Astrophysics and Space Science*, 2019.

[10] P. Krawczuk, S. Subramanya, G. Papadimitriou, R. Tanaka, and E. Deelman, "Galaxy Classification Workflow Implementation for the Pegasus Workflow Management System," Aug. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5297662

[11] T. Akiba *et al.*, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019.

[12] A. Jain *et al.*, "Lung Instance Segmentation Workflow Implementation for the Pegasus Workflow Management System," Aug. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5297479

[13] P. Krawczuk *et al.*, "Crisis Computing Workflow Implementation for the Pegasus Workflow Management System," Aug. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5298196

[14] F. Ofli, F. Alam, and M. Imran, "Analysis of social media data using multimodal deep learning for disaster response," 2020.

[15] F. Alam, F. Ofli, and M. Imran, "Crisismmd: Multimodal twitter datasets from natural disasters," in *Proceedings of the 12th International AAAI Conference on Web and Social Media (ICWSM)*, June 2018.

[16] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *In EMNLP*, 2014.

[17] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience," *Concurrency and computation: practice and experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[18] SciTech, "Pegasus panorama," https://github.com/pegasus-isi/pegasus/tree/panorama.

[19] K. Vahi *et al.*, "Rethinking data management for big data scientific workflows," in *Workshop on Big Data and Science: Infrastructure and Services*, 2013, funding Acknowledgments: OCI SDCI program grant #0722019 and OCI SI2-SSI program grant #1148515.

[20] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PloS one*, vol. 12, no. 5, 2017.

[21] G. Papadimitriou *et al.*, "End-to-end online performance data capture and analysis for scientific workflows," *Future Generation Computer Systems*, vol. 117, 2021, funding Acknowledgments: DOE DE-SC0012636.

[22] J. Liu *et al.*, "A survey of data-intensive scientific workflow management," *Journal of Grid Computing*, vol. 13, no. 4, 2015.

[23] M. Zaharia *et al.*, "Accelerating the machine learning lifecycle with mlflow." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[24] "Pachyderm," https://www.pachyderm.com, 2021.

[25] "Kubernetes," https://kubernetes.io, 2021.

[26] "Meet Michelangelo: Uber's Machine Learning Platform," Uber Engineering, 2017. [Online]. Available: https://eng.uber.com/michelangelo-machine-learning-platform/

[27] "Introducing FBLearner Flow: Facebook's AI backbone," Facebook Engineering, 2016. [Online]. Available: https://engineering.fb.com/2016/05/09/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/

[28] M. Zaharia *et al.*, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[29] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[30] S. Landset *et al.*, "A survey of open source tools for machine learning with big data in the hadoop ecosystem," *Journal of Big Data*, 2015.

[31] J. Carreira *et al.*, "Cirrus: A serverless framework for end-to-end ml workflows," in *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 13–24.

[32] M. S. Kurz, "Distributed double machine learning with a serverless architecture," in *Companion of the ACM/SPEC International Conference on Performance Engineering*, 2021, pp. 27–33.

[33] K. Zhou *et al.*, "A tool for top-down performance analysis of gpu-accelerated applications," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 415–416.

[34] "Nsight," https://developer.nvidia.com/nsight-graphics, 2021.

[35] "Pytorch," https://pytorch.org, 2021.

[36] "Tensorboard," https://www.tensorflow.org/tensorboard, 2021.

[37] M. Guignard *et al.*, "Performance characterization of state-of-the-art deep learning workloads on an ibm "minsky" platform," in *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.

[38] Z. Chishti *et al.*, "Memory system characterization of deep learning workloads," in *Proceedings of the International Symposium on Memory Systems*. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3357526.3357569

[39] F. Chowdhury *et al.*, "I/o characterization and performance evaluation of beegfs for deep learning," *Proceedings of the 48th International Conference on Parallel Processing*, 2019.