Exploration of Workflow Management Systems Emerging Features from Users Perspectives

Ryan Mitchell*, Loïc Pottier*, Steve Jacobs†, Rafael Ferreira da Silva*, Mats Rynge*, Karan Vahi*, Ewa Deelman*
* Information Sciences Institute, University of Southern California, Marina Del Rey, CA, USA
† National Ecological Observatory Network (NEON), Boulder, CO, USA
{rmitchel,lpottier,rafsilva,rynge,vahi,deelman}@isi.edu,
{sjacobs}@battelleecology.org

***Abstract*—There has been a recent emergence of new workflow applications focused on data analytics and machine learning. This emergence has precipitated a change in the workflow management landscape, causing the development of new data-oriented workflow management systems (WMSs) as opposed to the earlier standard of task-oriented WMSs. In this paper, we summarize three general workflow use-cases and explore the unique requirements of each use-case in order to understand how WMSs from both workflow management models (task-driven workflow management models and data-driven workflow management models) meet the requirements of each workflow use-case from the user's perspective. We analyze the applicability of the two main workflow models by carefully describing each model and by providing an examination of the different variations of WMSs that fall under the task-driven model. To illustrate the strengths and weaknesses of each workflow management model, we summarize the key features of four production-ready WMSs: Pegasus, Makeflow, Apache Airflow, and Pachyderm. Of these production-ready WMSs, three belong to the task-driven workflow management model (i.e., Pegasus, Makeflow, Apache Airflow) and one belongs to the data-driven workflow management model (i.e., Pachyderm). To deepen our analysis of the four WMSs examined in this paper, we implement three real-world use-cases to highlight the specifications and features of each WMS. The application of these real-world use-cases demonstrates how each workflow management model operates with the different applications. We present our final assessment of each WMS after considering the following factors: usability, performance, ease of deployment, and relevance. The purpose of this work is to offer insights from the user's perspective into the research challenges that WMSs currently face due to the evolving workflow landscape.**

***Index Terms*—Scientific workflow, Workflow Management System, Task-driven, Data-driven.**

I. Introduction

In the last two decades, scientific workflows have become mainstream thanks to their ability to empower scientific discoveries in virtually all fields of science [1]. During this time, key engineering challenges have been solved and a rich set of abstractions and interoperable software implementations have been developed [2]. These advancements have allowed scientists across various fields to begin reaping the benefits of workflow systems [3].

Traditionally, scientific workflows are described as directed-acyclic graphs (DAGs), in which nodes represent computational tasks and edges represent the dependencies of those tasks [4]. The traditional approach for orchestrating DAG-based workflows is to use task-based scheduling algorithms that spawn tasks for execution once their dependencies are satisfied. To keep up with the increased computing requirements, workflow systems have developed mechanisms to manage the distribution and execution of tasks on varied and across computing infrastructures such as local servers, campus computing clusters, high performance computing resources (such as XSEDE [5]), and even popular cloud computing platforms. These developments in workflow management are of primary concern to the field of scientific computing, where scientists often run complex pipelines that scale over hundreds and thousands of tasks [2].

There are four general workflow system use-cases that have been identified [3]:

- Traditional, scientific compute workflows (are both control-flow and data-flow driven);
- Data analytics workflows (including big data and machine learning);
- Sensor and Internet-of-Things (IoT) workflows; and
- Commercial, developer, and business-related workflows (are often control-flow driven).

One of the major trends among scientific applications recently concerns big data analytics and machine learning [4]. These data-oriented workflows pose different challenges when compared to traditional workflow structures, and they often require special features such as data provenance, data reproducibility, built-in functionality to easily and efficiently ingest data from cloud-based storage, and the ability to trigger data analytics pipelines as soon as new data are made available for processing.

A second type of data-oriented workflow that is gaining prominence in the scientific field is that of sensor-based workflows. Such workflows process data in a continuous fashion, with data being ingested in near real-time from distributed sources (e.g.,sensors that stream data to a central endpoint). To process such workflows, the ability to trigger computations based on the arrival of new data is paramount, in addition to the ability to replay processing on previous datasets if errors in processing are discovered post-mortem. These type of workflows are often control-flow driven, with each task

generally pushing results to a database.

Similar to sensor-based workflows, the rapid expansion of the Internet-of-Things (IoT) field, which now encompasses millions of interconnected devices, also raises the need for new workflow orchestration models [6]. In contrast to large-scale data analytics (which process large amounts of data in parallel), sensor-based and IoT-based workflows generally process smaller amounts of data at one time with a higher data arrival rate, placing a greater importance on new data as compared to old or late data.

Furthermore, in the developer and commercial communities, workflows are becoming increasingly important. Developers have started to automate more complex tasks such as unit tests and checking software build success. On the commercial side, as volumes of commercial data increase, businesses are turning to in-house workflow management systems to analyze their data. Companies have started to create a new generation of workflow management systems that are DAG oriented and are able to perform batch processing to their own specification (e.g., LinkedIn with Azkaban [7] to orchestrate Hadoop based pipelines, AirBnb with Apache Airflow [8], and Spotify with Luigi [9]). All of these systems are open source software, which benefits the scientific community in addition to the commercial space.

In contrast to scientific workflows, in the commercial field, workflows look very similar from one run to another. What matters here is: the ability to effectively monitor various runs of the same pipeline (represented as a DAG of tasks); the ability to compare and analyze the various runs via an intuitive user interface; and the option to automatically restart a task that failed. In addition to being used by companies for management or development purposes, these custom-developed solutions are also being used by scientists as building blocks [10], [11], enabling them to create their own light-weight workflow management solutions.

Based on the distinct characteristics and requirements of the different workflow use-cases described, we can broadly classify workflow management systems (WMSs) into two distinct categories: traditional, task-driven WMSs and modern, data-driven WMSs. In the traditional *task-driven* approach, workflow tasks are triggered for execution once all of their parent tasks have completed. In a more recent paradigm denoted as the *data-driven* approach, workflow task triggering mechanisms are steered by individual task data input and output. In this paper, we conduct a study on the key requirements and features that have driven the development of this new paradigm by exploring how workflow systems have addressed the recent challenges presented by these new workflow use-cases. We also identify open questions that have not yet been addressed by today's workflow management solutions. We first describe the paradigms in a WMS-agnostic and platform-agnostic manner, and we then present real world use-cases that have benefited from these state-of-the-art workflow system implementations. Note that we do not aim at performing a feature-by-feature comparison of workflow systems. Instead, our goal is to provide our own hands-on experience in dealing with such challenges from the WMS's and user's perspectives. It is also important to note that, during the last two decades, many of the authors of this paper have been involved in the scientific workflow community and have contributed to the development of workflow management systems (most notable of which is the Pegasus Workflow Management System [12]). Though Pegasus falls into the traditional, task-driven workflow management model, the authors of this paper are excited and intrigued by the new approaches and use-cases that have recently emerged.

For each use-case, we have tried to focus on representative workflow systems. We are aware that a plethora of workflow systems have been developed in the recent years, however it is not possible to account for every one of them. The purpose of this paper is to broadly classify use-cases and help the reader identify the workflow system that is best suited for their research needs.

More specifically, this work makes the following contributions:

1) We depict the differences between the traditional task-driven approach and the next-generation data-driven approach for workflow systems. We present several existing WMSs, along with their respective features, that fit each model. We also provide a detailed discussion about the new trends arising in the task-driven model.
2) We describe three real-world use-cases and explain how each of these use-cases fits into the different workflow management approaches (in terms of workflows and users requirements). We also present the potential difficulties that users may face when deploying these use-cases on different cyberinfrastructures.
3) We summarize our discussions, present our findings, and provide our insights about features that should be addressed by future WMSs in order to tackle next-generation scientific workflows.

This paper is organized as follows. In Section II, we provide an overview of the background and related work. Section III provides an overview of the requirements for both workflow management models by describing systems which implement each paradigm. In Section IV, we describe real-world use-cases for each paradigm executed using the four WMSs (Pegasus, Makeflow, Apache Airflow, and Pachyderm). Section V is dedicated to illustrating the experience from the perspective of both the user and the WMS. This is done by comparing the usability, performance, and relevancy of each use case. Finally, Section VI summarizes our findings about future scientific workflow management developments.

II. Background and Related Work

In this section, we aim to draw a broad picture of workflow solutions and designs, from scientific workflows to business workflows, and from academia to industrial workflows solutions. Barker et al. [3] characterize business workflows as control-flow and event oriented, while scientific workflows are more often data-flow oriented. Barker et al. also state that business workflows are static, meaning that they are defined

and implemented once, while scientific workflows usually change over time as new algorithms or scientific methods are being developed and improved.

A. Scientific workflows

One of the first models to represent a sequence of different computations is the directed acyclic graph (DAG) model [13]. In this *task-driven* approach, computational tasks, represented by nodes in the DAG, are the primary units. Many popular WMSs in the scientific community, such as Kepler [14], Makeflow [15], Taverna [16], and Pegasus [12] rely on a task-driven approach using a DAG representation. A DAG is a very simple and natural representation that allows WMSs to apply efficient scheduling [3] and data management optimizations.

Despite being less popular than the DAG representations, several graph models allowing cycles have also been developed. Mayer et al. [17] have proposed a workflow representation allowing iterations and loops. Cylc [18] is a Python workflow engine designed to manage cyclic workflows that would possibly run indefinitely by generating workflows at runtime during executions.

In the past, scientific workflows were traditionally compute-intensive but thanks to GPUs and new memory technologies, many data-intensive scientific workflows have been developed [4], [19]. In addition, from biology to astronomy, the number of scientific domains embracing workflow management systems is constantly growing [2]. The requirements and uses from one community to another, however, are not consistent [3].

Following these trends, new requirements have emerged in the scientific workflow user community. Among these requirements are: easy deployment on several cloud and HPC platforms and efficient data management with a strong data reproducibility aspect. To address these needs, workflow systems have evolved and adopted new technologies (such as containers support) to ensure better reproducibility and easier deployment. Zheng et al. [20] studied the deployment of a bioinformatics workflow with Makeflow [15] using Docker containers [21] and showed that fully-containerized execution leads to better reproducibility, easier deployment for the users, and resource usage improved by a factor of two.

Several new workflows systems have also adopted an API approach in which users programmatically define the workflow instead of giving it an abstract definition [9], [22]. An example of such a system is Parsl [22]. Parsl is a Python scripting workflow system that enables users to quickly define their workflows by directly annotating their python codes.

B. Data-oriented workflows

As described above, there have been many recent developments in the field of scientific workflows. With the community growing in size, more workflow management systems (WMS) are being developed in response to the community's need for specific features and evolving workflow paradigms. Driven by the popularity of data analytics and machine learning systems, these new WMSs are more data-oriented than their traditional counterparts.

Many studies on data-oriented workflow management have focused on the MapReduce [23] approach and its most known implementation: Apache Hadoop YARN [24]. Apache Hadoop YARN aims at decoupling resource management (e.g., scheduling, fault-tolerance) from the programming model. Apache Apex [25] is a data-oriented solution built on top of Hadoop and YARN that allows users to express both streaming and batch data pipelines with a DAG-based representation. It provides many data ingestion models such as IoT, social media, or sensors with Apache Kafka [26]. Apache Kafka [26] is a stream processing solution specifically designed to build real-time data pipelines. Besides MapReduce-based solutions, several generalist data-oriented solutions have been developed. Two examples of generalist data-oriented solutions are Pachyderm [27] and Nextflow [28].

Pachyderm [27] targets data versioning using a Git-based approach or using fully-containerized executions of data pipelines. In this study, we explain the features offered by Pachyderm in greater detail by leveraging the work presented by Novella et al. [29]. This work discusses Pachyderm's features and ease-of-use and explains how their bioinformatics use-case benefits from a fully-containerized system supporting data versioning. Nextflow [28] uses a domain-specific language that allows users to quickly prototype workflows running in containers. An interesting feature of Nextflow is the complete integration with several versioning platforms such as GitHub, which enables the workflow to check for updates and pull data from a given repository. On the machine learning (ML) side, Kubeflow [30], which is based on Kubernetes [21], has been designed to ease the management of ML workflows.

Besides science-agnostic WMSs like Pachyderm or Nextflow, several domain-specific solutions have also been developed, especially in the field of biology. Toil [31] is a WMS dedicated to large-scale biomedical data analysis. In Toil, each task runs in a Docker container and a Python API is provided to declare the workflow, thus following the model of Parsl [22]. Another example illustrating the need for specialized solutions is provided by Johnson et al. [32], which describes SABER/CONDUIT, a workflow system for neuroimaging that was developed to fulfill specific requirements according to the terms of neuroimaging that could not be satisfied by by agnostic-science solutions like Pachyderm [27] or even by specialized biomedical solutions like Toil [31].

C. Business workflows

In the commercial space, workflows are also becoming increasingly important. Companies need to analyze large volumes of data and are turning to workflow management systems as a result. To this end, companies such as LinkedIn and Yahoo have begun developing their own in-house WMSs to manage Hadoop jobs using Azkaban [7] and Oozie [33]. Additionally, a new industry has been created in which companies develop WMSs specifically for an enterprise as opposed to a general field. (e.g., Pachyderm [27]). Another industrial example from

the database side is Apache Cassandra [34], which is a NoSQL distributed database developed by Uber to manage online transactions processing at scale and to optimize their in-house data analytics workflows.

Many companies have also built their own in-house solutions to manage their developer and business workflows (i.e., nightly code checking, automating the deployment of containers to test newly developed features, processing and analyzing the significant amount of data they are facing). Airbnb has developed Airflow [8] to manage their own data and developer workflows, and Spotify has built Luigi [9] to handle their data analytics pipelines. Luigi is a Python module designed to help users run their complex pipelines by handling workflow scheduling, failure and retries. In contrast to Azkaban [7] and Oozie [33], Luigi is not only focused on Hadoop processing. With more companies developing their own workflow management solutions, the field of publicly-available WMSs has broadened significantly and now consists of many unique options with different tools, features and requirements.

In this paper, we attempt to distill these recent developments in the field of scientific workflows by presenting a basic comparison between the traditional paradigm of workflow management systems (such as Pegasus, Makeflow, and Apache Airflow) and the advent of newer systems intended for more specific purposes (such as Pachyderm). We also aim to aid users in the increasingly complex process of narrowing down which WMS or paradigm would fulfill their needs by examining a distinct set of representative workflow systems in a holistic manner.

III. Workflow Management Models

In this section, we present two workflow management system models, the task-driven and the data-driven models; and provide an overview of their respective requirements and key features. We also introduce different workflow management systems using those models.

A. Task-driven approach

***Traditional model*:** The task-driven approach has traditionally been based on and followed the principles of a directed acyclic graph, or DAG. The idea behind the task-driven model is to break a large and complex workflow into a sequence of individual computational tasks. A workflow management system using the task-driven model, usually defines computational tasks as the primary entity of work, and the workflow is defined as a graph, where nodes represent computational tasks and edges connecting nodes represent data and control dependencies (see Figure 1). A task can start its execution if, and only if, all of its predecessors have successfully completed. The task-driven model is simple to understand and well adapted to heavy computational tasks and is widely-used by HPC frameworks [35], as well as by numerous theoretical scheduling studies [36]. It is also very efficient, thanks to years of research focused on workflow optimization, scheduling strategies and data management, both from a practical and theoretical points of view. In addition, due to the decades of research and software development many task-driven WMS are mature and production-ready, such as Pegasus [12] or Makeflow [15]. This model allows users to target many different platforms, from large-scale HPC systems to distributed cloud platforms, grid infrastructures, or local clusters.

Recently, several new workflow management systems have been developed to bring more flexibility to the classic task-driven model with the ability to schedule a task or a sub-workflow at a given time (e.g., cron-based management), allow for conditional workflow execution, and improved error management mechanisms.

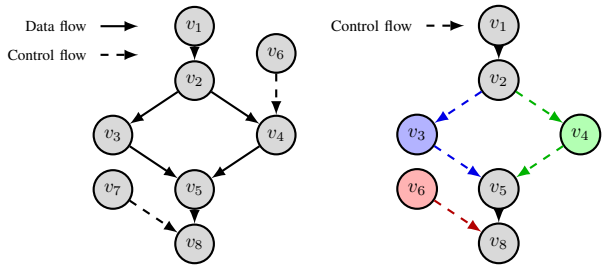
***Task triggering and conditional execution*:** One distinct feature of these next-generation task-driven workflow systems is the methods they use to trigger a workflow or a task. While traditional WMSs often rely on a manual triggering process, or the usage of the Linux utility `cron`, newer workflow systems have more innovative ways of triggering workflows. Several provide robust scheduling engines, and follow what we consider the "time-triggered" model, a subcategory of the task-driven model. For example, in Airflow [8], each DAG is associated with parameters such as a workflow's start date, an end date, a number of retries in case of failure, the delay between each retry, among others. A parameter called the *schedule interval* describes how many times a workflow will be scheduled between the start and the end dates, for example, daily, weekly, monthly, or any other `cron` expression. The scheduler runs in the background as a daemon and will pick up or kick off any DAG according to their start dates, end dates, and schedule intervals. Another interesting task triggering concept extending the possibilities of the task-driven model is the ability to trigger tasks without satisfying dependencies. In the time-triggered model, users are able to execute a task only if all its predecessors have failed, or if one predecessor failed (e.g., the red task in Figure 1(b)). This ability can be seen as an exception handling mechanism for workflow execution.

Finally, another major refinement when compared to the traditional task-driven approach is the support of "conditional execution", where a branch of the workflow is executed only if a given condition is satisfied (see Figure 1(b)).

***Workflow management systems*:** To evaluate the task-driven model, we consider three production WMSs, Pegasus [12], Makeflow [15], and Apache Airflow [8].

Pegasus [12] and Makeflow [15] are two well-established workflow management systems designed to manage and optimize the execution of large-scale scientific workflows on distributed resources, they provide container support on various commercial clouds such as AWS or Microsoft Azure, as well as HPC systems via support of various cluster job managers, including Slurm and PBS. An important difference between them is that, Pegasus allows control and data flow to express workflows while Makeflow allows only data flow.

Apache Airflow [8] is an Apache Software Foundation project, originally developed by Airbnb and released in 2016, that aims to provide a lightweight workflow management



(a) Classic representation of a DAG-based task-driven workflow with two types of dependencies.

(b) Workflow with a conditional execution in blue/green and a failed node in red.

Figure 1. Two examples of DAG-based task-driven workflows. On 1(b), if in v_2 a given condition is true then v_3 is executed, otherwise v_4 is spawned. In addition, the execution v_6 fails, but as v_5 successfully finished, v_8 is spawned.

solution to easily model, maintain, and monitor workflows. In contrast to the traditional task-driven model in which a DAG describes data and/or control exchanges, in Airflow a task is not supposed to exchange data with other tasks—they can only exchange metadata (i.e., only control flow) [8]. Airflow is very modular and provides many pre-built interfaces (*Hooks*) to common clouds and database systems such as Amazon S3, Google Cloud, or HDFS among others, and has a modular execution engine for computational tasks (*Operators*). Users are able to utilize multiple clouds on a single deployment. Airflow also supports containerized execution and orchestration through Kubernetes [21]. Note that Airflow is not a data streaming solution like Apache Spark [37].

B. Data-driven approach

With the advent of big data systems, machine-learning algorithms, and sensor-based workflows, novel data-driven workflow solutions have emerged. Compared to the previous approach, they provide better data provenance and versioning, better support for cloud-based storage, easier data ingestion, and a good scaling capability via the adoption of highly scalable orchestration solutions such as Kubernetes [21].

***Model*:** In this model, data are the primary units. A data-driven workflow can be represented as a DAG but instead of tasks being individual nodes, a node represents a *data repository* and the edges are the computational tasks. A data repository can be seen as a directory where data are structured as objects or files.

A task, usually denoted as a *pipeline* in data-driven terminology, describes the processing steps using the data in the incoming repository. A pipeline corresponds to the edge in the classic DAG representation (see Section III-B). Let v_i and v_j be two distinct data repositories, let $e_{i,j}$ be the edge from v_i to v_j. Then, v_i stores the input data used by the pipeline $e_{i,j}$ and v_j stores the output data produced by this pipeline. A successful pipeline will create a new repository for output files, this directory can be used as input directory for another pipeline, connecting pipelines to each other through repositories defines the workflow.

Typically, in data-oriented workflows, users would start processing the newer data as soon as they become available without to have to trigger the workflow themselves. Thus, the data-driven model is an *active* model, that continuously checks each data repository whether new data are ready to be processed. Notice that this model differentiates from stream-based workflows since computational tasks are not constantly running on computing nodes waiting for the next chunk of data to process.

***Reproducibility*:** As scientific workflows always use more and more data, the reproducibility of results and the data provenance become crucial. In the data-driven model, each data repository is versioned ensuring a complete data reproducibility and allowing users to execute workflows on each data version available. Note that, this feature does have a non-negligible cost in terms of storage. For example, Pachyderm uses a Git-inspired [38] data-versioning system, so users add data to repositories via a *commit* and these are then processed by the tasks (see Section III-B). All data processed by Pachyderm is versioned and stored, allowing the users to rollback and execute any pipelines on any data that have been versioned. Note that, Pachyderm also stores and versions the pipeline specifications. Coupled with a native containerized execution, this enable fully reproducible data pipelines.

***Cloud-based orchestration*:** Finally, an important feature often found in WMS following the data-driven model is the use of cloud-based container orchestration solutions such as Kubernetes [21], which allow efficient handling of large amounts of data in order to to have a reproducible and an easy deployment procedure on different cloud providers. In Pachyderm's case, each pipeline is contained in a Docker image and all pipelines managed through Kubernetes worker pods.

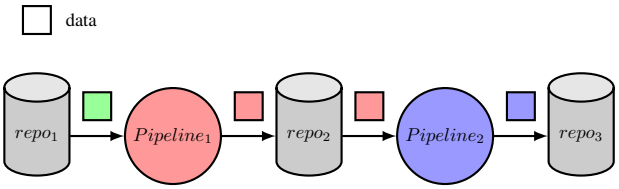


Figure 2. A simple example of a data-driven workflow. A piece of data is passing through different tasks, here called pipelines, that compute on the data.

***Workflow management system*:** Pachyderm [27], [29] aims to enable reproducible, collaborative, and scalable data science through a more innovative approach to workflow management. A Pachyderm workflow, or pipeline, is organized around data repositories (nodes in the DAG) containing files. Repositories are managed by the Pachyderm file system (PFS). Whereas Hadoop-based solutions are usually optimized for MapReduce processing, Pachyderm is data- and language-agnostic, meaning that Pachyderm is not limited to a given data format or programming language to process the data.

Using the active approach previously described, Pachyderm runs the pipeline on the input data and waits asynchronously

for new commits (i.e., new data to be processed by the pipeline). Pachyderm is built on top of numerous software layers and runs on top of widely-used commercial cloud providers (Amazon S3, Microsoft Azure, Google Cloud, etc.)

IV. Real-World Use Cases

To examine each workflow management paradigm and its associated workflow management systems, we chose three real-world workflow use-cases based on the four general workflow categories described in the introduction, choosing to omit the commercial and developer use-case as it is less pertinent to scientific workflows and interests. These real-world use-cases are as follows: (*i*) a traditional scientific compute workflow used to evaluate traditional task-driven WMSs Pegasus and Makeflow; (*ii*) a data streaming workflow to evaluate the newer-generation task-driven WMS Apache Airflow; and (*iii*) a sensor-based workflow to evaluate the data-driven WMS Pachyderm.

A. 1000 Genomes Workflow

The 1000 Genomes workflow is a traditional DAG-based bioinformatics workflow, fetching and parsing data from the 1000 Genomes Project [39]. The workflow aims to analyze mutational overlaps in humans, ultimately allowing statistical evaluation of potential disease-related mutations. The Project's Phase 3 and superpopulations data is downloaded and parsed (*Individuals* and *Populations* tasks), sorting amino acid substitutions and determining their potential phenotypic effects (*Sifting* tasks). Analysis is then performed in the *Frequency overlap* and *Pair overlap* tasks (see Figure 3).

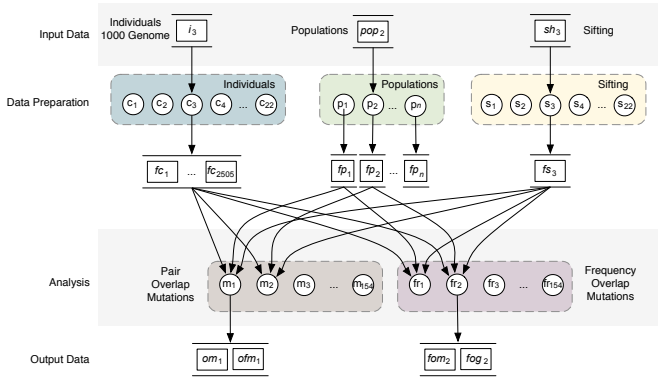


Figure 3. 1000 Genomes Workflow.

The workflow itself can be considered an example of a traditional or "classic" scientific DAG-based workflow because of its consistent dependencies and static nature. The workflow satisfies all DAG properties—each computational task in the workflow depends on the completion of previous parent tasks, and no changes to the workflow structure occurs. Additionally, the workflow is not dynamically triggered, i.e. it starts based on a user's command, and all workflow input data are known a priori. This workflow use-case has no particular requirements, only requiring data-flow dependencies and an available network connection to retrieve the dataset.

B. Streaming Data with CASA

Streaming data workflows have become increasingly popular in recent years. The University of Massachusetts' Collaborative Adaptive Sensing of the Atmosphere (CASA) project utilizes a sensor-based streaming data workflow for their Dallas/Fort Worth (DFW) weather radar testbed, which aggregates data from eight short-range weather radar sensors, providing higher data precision, accuracy and timeliness versus other, longer-range radar systems [40].

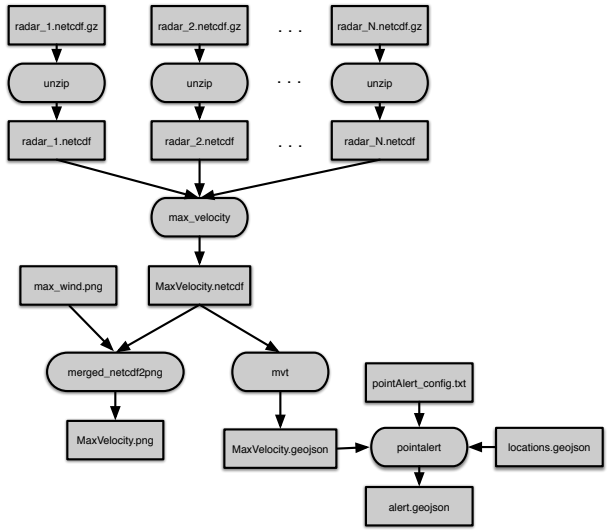


Figure 4. CASA Wind Velocity Workflow.

This paper focuses on a subsection of the CASA workflow pipeline [41], the calculation of maximum wind velocity and the sending of a customizable email alert to warn certain entities such as hospitals or airports of impending high wind velocity. The CASA workflow takes input data from each weather radar sensor (the `radar_N.netcdf.gz` files), unzips the data (*unzip* tasks), computes the maximum wind velocity around the DFW area (*max_velocity* task) and creates a graphical image of this data(*merged_netcdf2png* task). It then outputs velocity data `geojson` files (*mvt* task). Finally, these files are used to create high wind velocity alerts in the *pointalert* task (see Figure 4).

The CASA use-case has two basic requirements. Most importantly, the streaming property of the workflow requires workflow triggering every 75 seconds using new just-in-time data ingested since the last workflow run. Furthermore, computational executables are stored in a Docker container. While the use of the container is not required, its usage allows for easy portability and reproducibility.

C. Sensor-based Data-driven Workflow with NEON

The National Ecological Observatory Network (NEON) is an NSF open-science facility collecting ecological data from sensors across the US with the objective to study ecological processes and changes. NEON's instrument data pipeline takes raw sensor data from terrestrial and aquatic sensors and processes it for publication. Raw sensor data ranges from

resistance values and voltage at a low frequency of collection, to high frequency sonic anemometer data. The workflow converts this data into the appropriate unit of measurement using calibration coefficients, and performs QA/QC steps on the data to ensure the quality (see Figure 5). The workflow is a linear workflow that, as long as data is coming through the pipeline, gathers data from multiple sources (metadata, calibration data and raw sensor data) and process them to create significant output results.

An essential requirement of the NEON instrument data pipeline is the ability to reprocess data. This is necessary for several reasons, including improved algorithms, more recent calibration data, or late data. NEON desires to produce "null flags" for periods of time when data is not available, and if the data becomes available at a later date, it must be run through the processing pipeline.

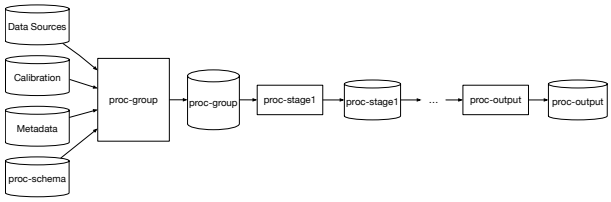


Figure 5. NEON processing workflow. The rectangles represent the pipelines and the cylinders the data repositories.

V. Holistic Evaluation

In this section, we present the results of a holistic evaluation of WMS for the models discussed above. We do not attempt to make a singular recommendation, rather this work aims to help users decide which workflow management model, either task-driven or data-driven fits their needs best, and provide an example WMS for each model to further demonstrate how a particular model can compliment a given workflow.

A. Experimental setup and criteria

In order to perform a thorough evaluation of each workflow management system, we used the following criteria:

1) *Setup and deployment:* For each WMS, we tested the installation process on a local cluster and using a cloud platform (AWS), using publicly-available documentation and user-facing support channels.
2) *Workflow implementation:* For each WMS, we examined the level of knowledge and effort required to model the relevant use-case and workflow.
3) *Workflow execution:* For each WMS, we researched features relating to workflow triggering, scalability, workflow resiliency, and how each WMS handles failures.
4) *Data management:* For each WMS, we studied how data is managed throughout the workflow execution, including whether data is transferred between computational tasks or workflows.

To test each WMS, two types of deployments were used in order to account for the majority of use-cases. First, initial WMS installation, testing, and workflow modeling was done on a local system. This deployment was used for initial testing as many WMS intended for scientific workflows are still deployed and used on local computers or clusters, and this use-case has its own unique challenges. Second, Airflow and Pachyderm, both considered cloud-ready WMS, were evaluated with special attention paid to cloud deployment, which presents separate challenges, and is increasingly important as some scientific users have begun transitioning their workflows from local to cloud deployments.

B. Traditional Task-driven WMS

Traditional task-driven WMS have been around for many years, and generally follow a strict DAG-based task-scheduling model. In our evaluation, we consider Pegasus and Makeflow as examples of traditional task-driven WMS.

***Pegasus*:** Pegasus is a powerful DAG-based workflow management system. Over the years, its functionality has been extended as user requirements have changed. Notable modern features include container support, several different workflow modeling APIs, and interactive workflow monitoring functionality. Ultimately, Pegasus is an excellent workflow management system for the 1000 Genome use-case, in part due to its Python workflow modeling API. The Python API allows the user the flexibility and ease-of-use of the Python language in creating a workflow pipeline.

In the case of the 1000 Genome use-case, the workflow uses a Python workflow generation script to enumerate files in the 1000 Genome dataset, and assign them as inputs of various tasks inside the workflow pipeline without knowing the exact file names, facilitating an easy workflow modeling process.

1) *Setup and deployment:* Pegasus installation is relatively simple due to its availability on different official repositories. Since Pegasus relies on HTCondor [42] as a task scheduler and interface to other cluster managers, additional effort is required to properly configure and describe the resources. Pegasus also supports cloud deployments on commercial cloud providers, and NSF-cloud infrastructures [41] such as Chameleon [43]. Since Pegasus was not natively designed for cloud support, cloud resources are still manually (or to some extent automatically) deployed in the cloud.
2) *Workflow implementation:* Pegasus provides a rich set of APIs (Python, Java, R, and Perl) for modeling workflows. These APIs provide a versatile mechanism for modeling large-scale workflows ($O(10^6)$ tasks), which is not often practical via graphical interfaces—though the entry barrier for non-expert users is higher.
3) *Workflow execution:* Pegasus is built for reliability and integrity, featuring several different types of workflow recovery methods, provenance data, and checkpointing abilities. Pegasus workflows also feature several "catalogs" listing the locations of key data resources, executable 'transformation' resources, and compute resources, allow-

ing for easy workflow portability where only resource configuration need to be changed.
4) *Data management:* Pegasus provides advanced mechanisms to efficiently manage data movement during workflow execution. During the workflow planning phase, Pegasus identifies data locations and augments the workflows with data transfer jobs for staging input and output data from/to storage resources. A wide range of protocols are supported, including access to cloud object storage, via Globus services, etc. Data throttling allows for increased throughput performance.

***Makeflow*:** Makeflow is another example of a traditional workflow management system. Whereas Pegasus targets flexible composition of DAG workflows via APIs with emphasis on functionality, Makeflow has a more rigid, yet simple, workflow modeling structure based on GNU `make`. In this case, dependencies between workflow tasks are automatically inferred from the data flow specified in the workflow description file, which alleviates the user's burden on defining task dependencies. On the other hand, defining control dependencies is not effortless, e.g. *zero* byte files could be created to mimic a control dependency. Such model also fulfills the task-driven requirements, and provides a relatively simple model for implementing the 1000 Genome workflow. Similarly to Pegasus, Makeflow also supports a wide range of computing platforms, including campus clusters, grids, HPC platforms, and clouds. It can also interface with HTCondor [42] and popular workload management interfaces via a workflow execution abstraction that translates compute tasks into the workload manager's jobs format.

1) *Setup and deployment:* Makeflow is released as part of the Cooperative Computing Tools, a self-contained set of tools for enabling large-scale computing. Due to its simple workflow structure model, its installation is relatively easy—though it is assumed a workload manager is already available (e.g., WorkQueue). Makeflow also supports AWS, where, for each task, it creates an EC2 VM, runs the task, and destroys the VM. Similarly to Pegasus, in Makeflow clouds are not fully-integrated.
2) *Workflow implementation:* Makeflow workflows are defined like 'Makefiles'. This structure is fairly simple for defining workflows where the data flow drives the tasks dependencies (entry barrier is reasonably low). The drawback of this approach is the limited flexibility for defining complex workflow patterns or control flows.
3) *Workflow execution:* Makeflow supports different workflow execution environments such as containers, and batch scheduler support, which are key for enabling large-scale executions. Makeflow also automatically retries failed tasks, but does not provide support for checkpointing or sophisticated error recovery methods.
4) *Data management:* In Makeflow, data is assumed to be directly accessible from the computing node (e.g., shared filesystem) or fetched from a remote source—support is limited to common Internet protocols. Notions of data provenance can be tracked back through the inherent structure of the Makeflow workflow model.

C. Recent task-driven WMS

One example of a more recent task-driven workflow management system is Airflow. Features such as extensible task support via *operators*, and a robust built-in workflow scheduling and triggering engine make the CASA use-case interesting for evaluating Airflow. Airflow's *operators* allow users to develop their own notions of what a workflow "task" means. While many traditional workflow systems limit computational tasks to executables called via the command line interface (CLI), or more recently, Python scripts, users have created operators for everything, from simple tasks such as sending an email or Slack message, to more complex tasks, such as running tasks inside a container, or managing a cloud deployment. The advantage to using an operator versus a comparable CLI tool can be seen in the operator's ease-of-use. The user has to simply learn how to use a preexisting operator, or write their own, instead of writing a custom CLI command and monitoring its various exit codes.

The workflow triggering engine is also extremely beneficial to modern workflows. More robust than a `cron`-based implementation on top of a traditional WMS, Airflow's scheduler allows many tasks to be run at custom times or intervals, and monitors them as such. This concept is extremely beneficial for the CASA workflow, which is started every 75 seconds in order to process new streaming data. Airflow's scheduler also has a concept of 'backfilling' a task, which allows users to tell Airflow to run a certain task while specifying the timeframe of the data being used.

1) Setup and deployment: Airflow is written on Python, which is the only required software prerequisite. Installing Airflow on a local cluster is relatively easy via the `pip` package-management. However, additional software such as the high-performance execution queue Celery might be needed to adapt Airflow to user's requirements. From a cloud perspective, Airflow primarily supports Google Cloud (GC), with extended support for Azure and AWS. Deployment on cloud platforms is simple, thanks to provided scripts (with a slight edge toward GC) but complex features such as cloud autoscaling still require configuration or external tools.
2) Workflow implementation: Through the extensive library of built-in operators, workflow implementation in Airflow is extremely flexible, and the Python API structure used to write workflows is easy to learn and understand.
3) Workflow execution: Airflow's scheduling features allow users a robust way to trigger their workflows. Airflow also has a built-in task retries parameter, like many other WMS and external software such as Apache Mesos allow checkpointing features.
4) Data management: Airflow features a basic way to pass data from task-to-task inside a workflow, but isn't as robust as other WMS's data management features. As such, tracking provenance is also more difficult.

D. Data-driven WMS

As described in the previous sections, Pachyderm is a prime example of a recently-developed WMS using the data-driven management paradigm. Written to allow portable data pipelines, with reproducibility and data provenance, Pachyderm is not explicitly designed for scientific workflows, but can be co-opted for scientific tasks. Pachyderm's workflow modeling process is interesting in that each task is defined individually as a data pipeline. The user has to specify not only what computational script, executable or transformation should be done, but which repository to use for input and output of data. Furthermore, Pachyderm requires usage of a container for each data pipeline. Multiple data pipelines can be 'joined' or 'split', using the same input or output repositories to manage data flow. Several pipelines, when combined together, can form a complete scientific workflow.

Pachyderm's container support is enabled via Kubernetes, with each data pipeline having its own Kubernetes pod. As Kubernetes allows for multiple containers to operate in a single pod, Pachyderm runs a sidecar container called '`storage`' that implements all the file retrieval and storage semantics that PFS uses. Pachyderm also injects a go binary into the user provided container ('`user`') that invokes the user's code as written in the pipeline specification. In this way Pachyderm runs the code in a loop when commits occur without requiring the user to add any special code to the docker container in question. An 'init' container runs before any of this happens, which creates the `/pfs` filesystem inside the pod that is used for all PFS operations.

One important feature of Pachyderm with regards to the NEON workflow pipeline is Pachyderm's robust data provenance tracking. The NEON project aims at publishing versioned data sets on an online portal to support open-science research. Although NEON can produce versions of these data sets without a solution like Pachyderm, Pachyderm's provenance tracking features allow NEON to inform the public what has changed between versions of the data sets, and why. NEON also needs to re-run their pipelines with field calibration data. For sensors that are calibrated regularly in the field this is inconvenient, as NEON has to reprocess when those calibrations are stored. For instruments that self-calibrate, and have no defined calibration period, this is borderline impossible without an on-demand reprocessing capability. Pachyderm allows NEON to do this, by triggering only the pieces of the pipeline necessary to run when a change comes in via the commit system.

1) Setup and deployment: Not differently from other WMS, Pachyderm requires several dependencies, including Kubernetes, which itself requires expert knowledge (as for HTCondor). Pachyderm is intended for cloud deployments, and several utilities exist to automate installation on cloud platforms like AWS. However, Pachyderm has limited local cluster deployment, requiring a Kubernetes cluster and S3-compatible storage. Compared to Airflow, which allows more fine-grain resource management, Pachyderm is more restrictive—once Pachyderm is deployed on a given cloud, all tasks execute on this cloud.
2) Workflow implementation: Because of Pachyderm's unique commit-based pipeline triggering, data pipelines must output files, and file and folder structures inside the PFS repositories might need to be organized for optimal data flow. However commit-base triggering features provided by Pachyderm are potentially essential scientific workflows, as highlighted with the NEON use-case.
3) Workflow execution: Pachyderm has the ability to scale over a Kubernetes cluster, which is extremely simple. Pachyderm automatically retries each data pipeline based upon task exit code. However, for certain workflows, error management might not be as easy as other WMS, since actual workflows are buried in containers inside Kubernetes pods.
4) Data management: Data passing is a strong point of Pachyderm, with data flow and provenance being one of the headline features of the WMS. Provenance also can be tracked back through a workflow, with PFS repositories tracking files and their respective commits.

VI. Conclusion

This work's main objective is to describe interesting trends and concepts in next-generation scientific and commercial workflow management systems, from a user's perspective. To this end, we analyzed how two different workflow management paradigms, namely the task-driven and data-driven paradigms, can be applied to real-world use-cases.

From the four generic use-cases detailed in the introduction, we carefully described three real-world use-cases. With a traditional scientific workflow with the 1000 Genome Project workflow, and two different sensor-based workflows with the CASA and NEON soil workflows, we presented a cross-section of traditional and next-generation workflows to evaluate trends and developments in the workflow space.

For the task-driven and data-driven paradigms, we selected representative workflow management systems. Pegasus, Makeflow and Apache Airflow represent the task-driven model, while Pachyderm represents the data-driven model. Each model is thoroughly and holistically evaluated, along with their associated workflow management systems. While performing this evaluation, we highlighted the rise in new technologies and innovations, including containers and the cloud, and new workflow management use-cases, such as big data analytics, large-scale science and machine learning. We then examined how these changes impact future workflows and workflow management system development, and discussed how possible benefits can be applied to established WMS solutions. Using several real-world use-cases, we highlighted how each WMS's unique features can be an asset to certain next-generation workflows, and emphasized how these features set each WMS apart from one another.

Future work consists of exploring more real-world use-cases such as IoT workflows or large-scale data analytics, as well as

more WMS solutions and approaches, e.g. Apache Kafka for real-time data streaming. We would also like to evaluate how techniques developed by these next-generation WMS could benefit to traditional scientific workflows.

Acknowledgments. This work is funded by NSF contract #1842042: "Pilot Study for a Cyberinfrastructure Center of Excellence". The National Ecological Observatory Network is a program sponsored by the National Science Foundation and operated under cooperative agreement by Battelle Memorial Institute. This material is based in part upon work supported by the National Science Foundation through the NEON Program.

REFERENCES

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

[2] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. V. Hemert, "Scientific workflows: Moving across paradigms," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, p. 66, 2017.

[3] A. Barker and J. Van Hemert, "Scientific workflow: a survey and research directions," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2007, pp. 746–753.

[4] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *Journal of Grid Computing*, vol. 13, no. 4, pp. 457–493, 2015.

[5] A. Rouhani, E. Bernhardsson, and E. Freider. Extreme science and engineering discovery environment (XSEDE). [Online]. Available: http://www.xsede.org/

[6] M. Nardelli, S. Nastic, S. Dustdar, M. Villari, and R. Ranjan, "Osmotic flow: Osmotic computing+ iot workflow," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 68–75, 2017.

[7] R. Sumbaly, J. Kreps, and S. Shah, "The big data ecosystem at linkedin," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 1125–1134.

[8] M. Beauchemin. (2014) Apache Airflow Project. [Online]. Available: https://airflow.incubator.apache.org/

[9] A. Rouhani, E. Bernhardsson, and E. Freider. (2012) Apache Luigi Project. [Online]. Available: https://github.com/spotify/luigi

[10] A. Mechev, R. Oonk, T. Shimwell, A. Plaat, H. Intema, and H. Rottgerin, "Fast and reproducible lofar workflows with aglow," in *2018 IEEE 14th International Conference on e-Science (e-Science)*. IEEE, 2018, pp. 136–144.

[11] M. Kotliar, A. Kartashov, and A. Barski, "Cwl-airflow: a lightweight pipeline manager supporting common workflow language," *bioRxiv*, p. 249243, 2018.

[12] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[13] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Annals of discrete mathematics*. Elsevier, 1979, vol. 5, pp. 287–326.

[14] B. Ludäscher, I. Altintas, C. Berkley *et al.*, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[15] M. Albrecht, P. Donnelly, P. Bui, and D. Thain, "Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids," in *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*. ACM, 2012, p. 1.

[16] T. Oinn, M. Addis, J. Ferris *et al.*, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[17] A. Mayer, S. McGough, N. Furmento, W. Lee, S. Newhouse, and J. Darlington, "Iceni dataflow and workflow: Composition and scheduling in space and time," in *UK e-Science All Hands Meeting*, vol. 634. Nottingham, UK, 2003, p. 627.

[18] H. Oliver, M. Shin, D. Matthews *et al.*, "Workflow automation for cycling systems: The cylc workflow engine," *Computing in Science & Engineering*, 2019.

[19] J. S. Vetter, R. Brightwell *et al.*, "Extreme Heterogeneity 2018 - Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 12 2018.

[20] C. Zheng, B. Tovar, and D. Thain, "Deploying high throughput scientific workflows on container schedulers with makeflow and mesos," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 130–139.

[21] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[22] Y. N. Babuji, K. Chard, I. T. Foster, D. S. Katz, M. Wilde, A. Woodard, and J. M. Wozniak, "Parsl: Scalable parallel scripting in python." in *IWSG*, 2018.

[23] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[24] V. K. Vavilapalli, A. C. Murthy, C. Douglas *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[25] H. Pathak, M. Rathi, and A. Parekh, "Introduction to real-time processing in apache apex," *Int. J. Res. Advent Technol.*, p. 19, 2016.

[26] N. Garg, *Apache Kafka*. Packt Publishing Ltd, 2013.

[27] Pachyderm, Inc. (2017) Pachyderm. [Online]. Available: https://www.pachyderm.io/

[28] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature biotechnology*, vol. 35, no. 4, p. 316, 2017.

[29] J. A. Novella *et al.*, "Container-based bioinformatics with pachyderm," *Bioinformatics*, vol. 35, no. 5, pp. 839–846, 2018.

[30] Kubeflow. [Online]. Available: https://www.kubeflow.org/

[31] J. V. et al., "Toil enables reproducible, open source, big biomedical data analyses," *Nature Biotechnology*, vol. 35, no. 4, pp. 314–316, 2017.

[32] E. C. Johnson, M. Wilt, L. M. Rodriguez, R. Norman-Tenazas *et al.*, "Toward a reproducible, scalable framework for processing large neuroimaging datasets," *BioRxiv*, p. 615161, 2019.

[33] M. Islam *et al.*, "Oozie: towards a scalable workflow management system for hadoop," in *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*. ACM, 2012, p. 4.

[34] A. Cassandra, "Apache cassandra," *Website. Available online at http://planetcassandra. org/what-is-apache-cassandra*, p. 13, 2014.

[35] G. Bosilca *et al.*, "Dague: A generic distributed dag engine for high performance computing," *Parallel Computing*, vol. 38, no. 1-2, pp. 37–51, 2012.

[36] R. Sethi, "Scheduling graphs on two processors," *SIAM Journal on Computing*, vol. 5, no. 1, pp. 73–82, 1976.

[37] M. Zaharia, R. S. Xin, P. Wendell *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[38] L. Torvalds and J. Hamano. (2005) Git: Fast version control system. [Online]. Available: http://git-scm.com

[39] R. Ferreira da Silva, R. Filgueira, E. Deelman, E. Pairo-Castineira, I. M. Overton, and M. Atkinson, "Using simple pid-inspired controllers for online resilient resource management of distributed scientific workflows," *Future Generation Computer Systems*, vol. 95, pp. 615–628, 2019.

[40] H. M. N. Dilum Bandara, A. P. Jayasuman, and M. Zink, "Radar networking in collaborative adaptive sensing of atmosphere: State of the art and research challenges," in *2012 IEEE Globecom Workshops*, Dec 2012, pp. 1378–1383.

[41] E. Lyons, G. Papadimitriou, C. Wang, K. Thareja, P. Ruth, J. Villalobos, I. Rodero, E. Deelman, M. Zink, and A. Mandal, "Toward a dynamic network-centric distributed cloud platform for scientific workflows: A case study for adaptive weather sensing," in *15th eScience Conference*, 2019, funding Acknowledgments: NSF 1826997.

[42] 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.

[43] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, J. Manbretti, P. Rad, and R. Paul, "Chameleon: a scalable production testbed for computer science research," *Contemporary High Performance Computing*, vol. 3, 2017.