# WfCommons: A Framework for Enabling Scientific Workflow Research and Development

Tainã Coleman[a,c,*], Henri Casanova[b], Loïc Pottier[a], Manav Kaushik[c], Ewa Deelman[a,c], Rafael Ferreira da Silva[a,c,*]

[a] *University of Southern California, Information Sciences Institute, Marina del Rey, CA, USA*
[b] *Information and Computer Sciences, University of Hawaii, Honolulu, HI, USA*
[c] *University of Southern California, Department of Computer Science, Los Angeles, CA, USA*

**Abstract**

Scientific workflows are a cornerstone of modern scientific computing. They are used to describe complex computational applications that require efficient and robust management of large volumes of data, which are typically stored/processed on heterogeneous, distributed resources. The workflow research and development community has employed a number of methods for the quantitative evaluation of existing and novel workflow algorithms and systems. In particular, a common approach is to simulate workflow executions. In previous works, we have presented a collection of tools that have been adopted by the community for conducting workflow research. Despite their popularity, they suffer from several shortcomings that prevent easy adoption, maintenance, and consistency with the evolving structures and computational requirements of production workflows. In this work, we present *WfCommons*, a framework that provides a collection of tools for analyzing workflow executions, for producing generators of synthetic workflows, and for simulating workflow executions. We demonstrate the realism of the generated synthetic workflows by comparing their simulated executions to real workflow executions. We also contrast these results with results obtained when using the previously available collection of tools. We find that the workflow generators that are automatically constructed by our framework not only generate representative same-scale workflows (i.e., with structures and task characteristics distributions that resemble those observed in real-world workflows), but also do so at scales larger than that of available real-world workflows. Finally, we conduct a case study to demonstrate the usefulness of our framework for estimating the energy consumption of large-scale workflow executions.

*Keywords:* Scientific Workflows, Workflow Management Systems, Simulation, Distributed Computing, Workflow Instances

## 1. Introduction

Scientific workflows are relied upon by thousands of researchers [1] for managing data analyses, simulations, and other computations in almost every scientific domain [2]. Scientific workflows have underpinned some of the most significant discoveries of the last decade [3, 4]. These discoveries are in part a result of decades of Workflow Management System (WMS) research, development, and community engagement to support the sciences [5, 6]. As workflows continue to be adopted by scientific projects and user communities, they are becoming more complex and require more sophisticated workflow management capabilities. Workflows are being designed that can analyze terabyte-scale datasets, be composed of millions of individual tasks that execute for milliseconds up to several hours, process data streams, and process static data in object stores. Catering to these workflow features and demands requires WMS research and development at several levels, from algorithms and systems all the way to user interfaces.

A traditional approach for testing, evaluating, and evolving WMSs is to use full-fledged software stacks to execute applications on distributed platforms and testbeds. Although seemingly natural, this approach has severe shortcomings including lack of reproducible results, limited platform configurations, and time and operational costs. An alternative that reduces these shortcomings is to use simulation, i.e., implement and use a software artifact that models the functional and performance behaviors of software and hardware stacks of interest. Thus, the scientific workflow community has leveraged simulation for the development and evaluation of, for example, novel algorithms for scheduling, resource provisioning, and energy-efficiency, workflow data footprint constraints, exploration of data placement strategies, among others [7, 8, 9].

Studying the execution of workflows in simulation requires sets of workflow application instances to be used as benchmarks. This is so that quantitative results are obtained for a range of representative workflows. In [10], we have described a collection of tools and data that together have enabled research and development of the Pegasus WMS [3], and have also been used extensively by the workflow community [11, 12, 13, 14, 15][1]. Despite the popularity of this pioneer effort, it lacks (*i*) a common format for representing a workflow's execution

*Corresponding address: USC Information Sciences Institute, 4676 Admiralty Way Suite 1001, Marina del Rey, CA, USA, 90292

*Email addresses:* tcoleman@isi.edu (Tainã Coleman), henric@hawaii.edu (Henri Casanova), lpottier@isi.edu (Loïc Pottier), manavkau@usc.edu (Manav Kaushik), deelman@isi.edu (Ewa Deelman), rafsilva@isi.edu (Rafael Ferreira da Silva)

[1] And also as seen by the high number of citations on Google Scholar.

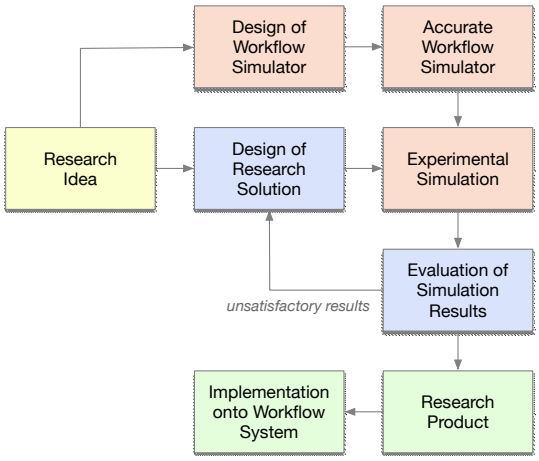



Figure 1: Simulation-driven engineering life cycle (adapted from [16]).

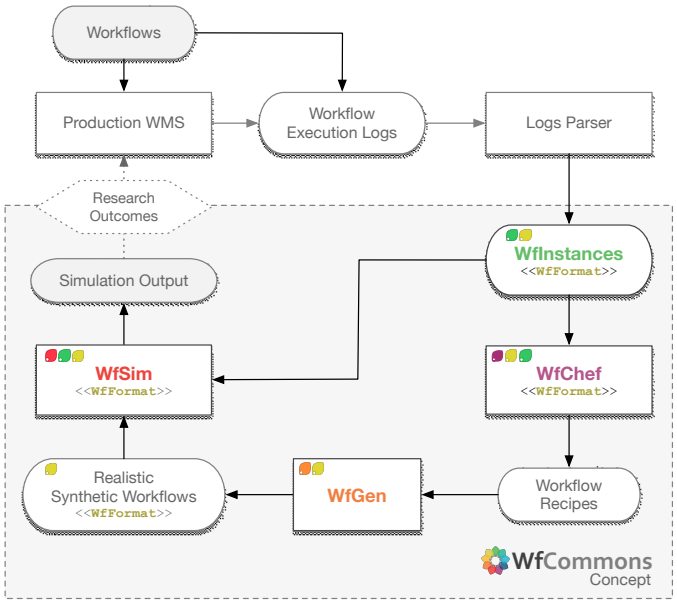



Figure 2: The WfCommons conceptual architecture.

in a way that is agnostic to WMS that was used to execute it; (*ii*) methods for deriving workflow structure and workflow task performance characteristics based on workflow execution; and (*iii*) techniques for automating the process of producing generators of realistic synthetic workflows for any given workflow application.

In this paper, we present ***WfCommons*** [17], an open-source framework that aims at supporting and at bridging theoretical and practical aspects of workflow systems research and development. The broad objective of WfCommons is to enable simulation-driven engineering of workflow systems (Figure 1) [16]. As such, it must address three main technical challenges: (i) the gathering and archiving of real-world workflow instances from diverse application domains based on their executions using diverse workflow systems; (ii) the analysis of real-world workflows so as to understand their fundamental structures and automatically generate representative synthetic instances at arbitrary scales; (iii) the accurate and scalable simulation of workflow executions on arbitrary platforms. WfCommons addresses these challenges via several mechanisms and techniques implemented as part of usable tools, which remedy the shortcomings of our previous set of tools. Specifically, WfCommons uses a system-agnostic JSON format for representing workflow instances based on execution logs. WfCommons also provides an open-source Python package to analyze workflow instances and produce generators of realistic synthetic workflow instances, generated in that same format. Workflow simulators that support this format can then take real-world and synthetic workflow instances as input for driving the simulation. Figure 2 shows an overview of the WfCommons conceptual architecture. Information in workflow execution logs is extracted as **workflow instances** represented using the common JSON format. Workflow "recipes" are obtained from the analysis of sets of workflow instances for a particular application. More precisely, a recipe embodies results from statistical analysis and distribution fitting performed for each workflow task type so as to characterize task runtime and input/output data sizes. The recipes also incorporates information regarding the graph structure of the workflows (tasks dependencies and frequency of occurrences), which are automatically derived from the analysis of the workflow instances. Each recipe is then used for automatically producing a **workflow generator**, which in turn produces synthetic workflow instances that are representative of the application domain. Finally, these instances can be used by a **workflow simulator** for conducting experimental workflow research and development. Specifically, this work makes the following contributions [2]:

1. A collection of workflow execution instances acquired from actual executions of state-of-the-art compute- and data-intensive workflows in a cloud environment;
2. A common format for representing both collected instances and generated synthetic instances;
3. An open source Python package [19] that provides methods for analyzing instances, deriving recipes, and generating representative synthetic instances;
4. A collection of open-source WMS simulators and simulation frameworks that support our common format;
5. An evaluation of the accuracy of WfCommons' generated synthetic workflows and a comparison to our previous sets of tools [10, 18];
6. A case study in which we demonstrate the usefulness of WfCommons in the context of energy-efficient workflow executions.

This paper is organized as follows. Section 2 discusses related work. The WfCommons project and the associated concepts and tools are explained in Section 3. Section 4 provides an experimental evaluation of the accuracy of WfCommons' generated synthetic workflows. Section 5 presents our case-study on energy-efficient workflow executions. Finally, Section 6 concludes with a summary of results and perspectives on future work.

[2] A short version of this work was previously published in [18].

## 2. Related Work

Workload archives are widely used for distributed computing research, to validate assumptions, to derive workload models, and to perform experiments by replaying workload executions, either in simulation or on real-world platforms. Available repositories, such as the Parallel Workloads Archive [20], the Grid Workloads Archive [21], and the Failure Trace Archive [22], contain data acquired at the infrastructure level at compute sites, or by monitoring or obtaining logs from deployed compute services. The workloads in these archives do include data generated by workflow executions. However, the information captured is about individual job executions and about resource utilization. As a result, there is at best little information on the task dependency structure of workflows.

In the context of scientific workflows, the Common Workflow Language (CWL) [23] is an open standard for describing workflows in a way that makes them portable and scalable across a variety of software and hardware environments. Our proposed common format (described below) is conceptually similar to the CWL standard, though our format captures performance metrics data (e.g., volumes of I/O reads and writes, runtime, power consumption, etc.) and compute resource characteristics, which are key for generating realistic workflow instances. The recently established Workflow Trace Archive [24] is an open-access archive that provides a collection of execution instances from diverse computing infrastructures and tools for parsing, validating, and analyzing instances. To date, the archive has collected instances from 11 existing online repositories (including 10 instances obtained from a preliminary version of WfCommons) and uses an object-oriented representation (based on the Parquet columnar storage format used in Hadoop) for documenting instances. Our format instead uses JSON, which is agnostic to the programming language used for processing instances. Also, the format used in [24] captures workflow executions information in terms of resource usage on the specific hardware platform used to execute the workflow. As a result, it is difficult to use this information to reconstruct a platform-independent, abstract workflow structure. By contrast, while WfCommons also records platform-specific behaviors in its instances, in addition it ensures that the abstract workflow structure is directly available from these instances. This is crucial for research purposes, as abstract workflow structures are needed for, for instance, simulating workflow executions on platform configurations that differ from that used to collect the workflow execution instance.

Several studies have used synthetic workflows to explore how different workflow features impact execution and interplay with each other (e.g., number of tasks, task dependency structure, task execution times). Tools such as SDAG [25] and DAGGEN [26] generate random workflow instances based on the number of tasks, the maximum number of levels that can be spanned by an edge, the edge density, the data-to-computation ratio, the width, etc. DAGEN [27] generates random DAGs for parallel programs modeled according real instances of parallel programs with respect to task computation and communication payloads. DAGITIZER [28], an extension of DAGEN-A, is applicable to grid workflows in which all parameters are generated randomly. Although these generators can produce a very diverse set of DAGs, they may not necessarily be representative of real-world scientific workflows.

Using the structure of real-world workflow instances to generate DAGs for specific applications is an alternative to the random generation approach. The work in [29] identifies workflow "motifs", or sub-structures, that are used to reverse engineer workflow structures based on the data created and used by the tasks. Although these motifs allow for automated workflow generation, identifying them is an arduous manual process. The work in [30] targets business process requirements such as parallelism, choice, synchronization, etc., and identifies over forty workflow patterns. These patterns can be mapped to structures in real scientific workflows [31], but they do not necessarily respect the ratios of the different types of tasks. The problem lies in the fact that a workflow structure is not only defined by a set of vertices and edges, but also by the task type – i.e. executable name, of each vertex.

In [32], application skeletons are used to build synthetic workflows that represent real applications for benchmarking. In our previous work [10], we developed a tool for generating synthetic workflow configurations based on real-world workflow instances. As a result, the overall structure of generated workflows was reasonably representative of real-world workflows. But that tool uses only two types of statistical distributions (uniform and normal), and as a result workflow performance behavior may not be representative (see results in Section 4).

## 3. The WfCommons Framework

The WfCommons project (`https://wfcommons.org`) is an open source framework for enabling scientific workflow research and development. It provides foundational tools for analyzing workflow execution instances, and generating synthetic, yet realistic, workflow instances. These instances can then be used for experimental evaluation and development of novel algorithms and systems for overcoming the challenge of achieving efficient and robust execution of ever-demanding workflows on increasingly complex distributed infrastructures.

Figure 2 shows an overview of the research and development life cycle that integrates the four major components WfCommons: (i) workflow execution instances (WfInstances), (ii) workflow recipes (WfChef), (iii) workflow generator (WfGen), and (iv) workflow simulator (WfSim).

### *3.1. WfInstances*

Catalogs of workflow instances are instrumental for evaluating workflow solutions in simulation or in real conditions. The WfInstances component targets the collection and curation of open-access production workflow instances from various scientific applications, all made available using a common format. A workflow instance is built based on logs of an actual execution of a scientific workflow on a distributed platform (e.g., clouds, grids, clusters) using a WMS. Specifically, the three main types of information included in the instance are:

Table 1: Collection of workflow execution instances hosted in WfInstances. All instances were obtained using the Pegasus and Makeflow WMSs running on the Chameleon cloud platform.

| Application | WMS | Science Domain | Category | # instances | # Tasks | Runtime and Input/Output Data Sizes Distributions |
|---|---|---|---|---|---|---|
| 1000Genome | Pegasus | Bioinformatics | Data-intensive | 22 | 8,844 | alpha, chi2, fisk, levy, skewnorm, trapz |
| BLAST | Makeflow | Bioinformatics | Compute-intensive | 15 | 2,245 | arcsine, argus, trapz |
| BWA | Makeflow | Bioinformatics | Data-intensive | 15 | 10,560 | arcsine, argus, rdist, trapz |
| Cycles | Pegasus | Agroecosystem | Compute-intensive | 24 | 30,720 | alpha, beta, chi, chi2, cosine, fisk, levy, pareto, rdist, skewnorm, triang |
| Epigenomics | Pegasus | Bioinformatics | Data-intensive | 26 | 15,242 | alpha, beta, chi2, fisk, levy, trapz, wald |
| Montage | Pegasus | Astronomy | Compute-intensive | 17 | 37,619 | alpha, beta, chi, chi2, cosine, fisk, levy, pareto, rdist, skewnorm, wald |
| Seismology | Pegasus | Seismology | Data-intensive | 11 | 6,611 | alpha, argus, fisk, levy |
| SoyKB | Pegasus | Bioinformatics | Data-intensive | 10 | 3,360 | argus, dweibull, fisk, gamma, levy, rayleigh, skewnorm, triang, trapz, uniform |
| SRA Search | Pegasus | Bioinformatics | Data-intensive | 25 | 1,580 | arcsine, argus, beta, dgamma, fisk, norm, rdist, trapz |
| 9 applications | 2 WMSs | 4 domains | 2 categories | 165 | 116,781 | 21 probability distributions |

- Workflow task execution metrics (runtime, input and output data sizes, memory used, energy consumed, CPU utilization, compute resource that was used to execute the task, etc.);
- Workflow structure information (inter-task control and data dependencies); and
- Compute resource characteristics (CPU speed, available RAM, etc.).

***Workflow Instance Format –*** The WfCommons project uses a common format, **WfFormat**, for representing collected workflow instances and generated synthetic workflows instances. Workflow simulators and simulation frameworks that support WfFormat can then use both types of instances interchangeably. WfFormat uses a JSON specification (available on GitHub [33]), which captures all relevant instance information as listed above. The GitHub repository also provides a Python-based JSON schema validator for verifying the syntax of JSON instance files, as well as their semantics, e.g., whether all files and task dependencies are consistent. Users are encouraged to contribute additional workflow instances for any scientific domain, as long as they conform to WfFormat. Currently, WfCommons provides parsers for converting execution logs into WfFormat for two state-of-the-art WMSs: Pegasus [3] and Makeflow [34].

***Collected Workflow Instances –*** An integral objective of the WfCommons project is to collect and reference open access workflow instances from production workflow systems. Table 1 summarizes the set of workflow instances currently hosted in WfInstances. These instances are from nine representative science domain applications supported by Pegasus or Makeflow, for workflows composed of compute- and/or data-intensive tasks. (Note that although a workflow may be categorized overall as, for example, data-intensive, it may include CPU-intensive tasks.) We argue that the 165 archived workflow instances form a representative set of small- and large-scale workflow configurations [35, 36, 37, 38]. For instance, Montage workflow instances can be composed of tens of thousands of short-running CPU-intensive tasks [35], while 1000Genome workflow instances comprise a few hundreds data-intensive tasks that operate over 40 GB of data [36]. BWA workflow instances comprise a few thousands short-running data-intensive tasks that process O(100) MB each [37]. Although Cycles workflow instances are mainly composed of compute-intensive tasks, they are also comprised of a small subset of data-intensive tasks that operates over 10 GB of data [38]. In addition to consuming/producing large volumes of data processed by thousands of compute tasks, the structures of these workflows are sufficiently complex and heterogeneous to encompass current and emerging large-scale workflow execution patterns [39].

### *3.2. WfChef*

WfChef is the WfCommons component that automates the construction of synthetic workflow generators for any given workflow application. The input to this component is a set of real workflow instances described in the *WfFormat* (e.g., instances available in *WfInstances*). WfChef automatically analyzes the real workflow instances for two purposes. First, it discovers workflow subgraphs that represent fundamental task dependency patterns. Second, it derives statistical models of the workflow tasks' performance characteristics. WfChef then outputs a "recipe", that is, a data structure that encodes the discovered pattern occurrences as well as the statistical models of workflow task characteristics. This recipe is then used by *WfGen* (see Section 3.3) to generate realistic synthetic workflow instances with any arbitrary number of tasks. The way in which WfChef operates is depicted in the top part of Figure 4, and hereafter we provide a brief overview of the methods used to construct recipes. A detailed description of the algorithms and an evaluation of their accuracy is provided in [40].

***Finding Pattern Occurrences –*** Given a set of real workflow instances for an application, WfChef finds the patterns that have more than one occurrence in the same workflow graphs and records these pattern occurrences. To identify patterns, WfChef defines a task's **type** as the kind of computation performed by the task, which is given by the task's name as extracted from the workflow execution logs (e.g., the name of the executable that is invoked to perform the computation). WfChef then recursively computes a unique ID for each task, called the **type hash**, based on the task's type and the type hashes of all the task's ancestors and descendants. The type hash of a task thus encodes information about the graph's structure and the role of the task in that structure. Finally, the type hash of a sub-graph is defined as the set of the type hashes of the tasks in that sub-graph. WfChef defines a (repeating) pattern as a set of disjoint sub-graphs that occur in a workflow instance's graph that have all the same type hash (i.e., a pattern is an equivalence class of the set of sub-graphs, where equivalence is defined as type hash

```
"individuals": {
    "runtime": {
        "min": 48.846,
        "max": 192.232,
        "distribution": {
            "name": "skewnorm",
            "params": [
                11115267.652937062,
                -2.9628504044929433e-05,
                56.03957070238482
            ]
        }
    },
    ...
}
```

Listing 1: Example of an analysis summary showing the best fit probability distribution for runtime of the `individuals` tasks in 1000Genome workflows.

equality). Two sub-graphs with the same type hash, i.e., with tasks of the same types and same dependency structures, do not necessarily have the same number of tasks but are occurrences of the same pattern.

To find pattern occurrences, WfChef proceeds as follows:

1. Pick two distinct tasks, $t_1$ and $t_2$, with *the same type hash*;
2. $S_1 = \{t_1\}$ and $S_2 = \{t_2\}$.
3. Add to $S_1$, resp. $S_2$, all the parents and children of tasks in $S_1$, reps. $S_2$;
4. $S_1 = S_1 - S_1 \cap S_2$; $S_2 = S_2 - S_1 \cap S_2$;
5. If $S_1$ or $S_2$ has increased in size after steps 3 and 4, go back to step 3;
6. Otherwise, $S_1$ and $S_2$ are identified as two occurrences of the same pattern.

Note that because $t_1$ and $t_2$ have the same type hash, it is guaranteed that $S_1$ and $S_2$ will have the same type hash, and thus are occurrences of the same pattern.

***Modeling Performance Characteristics –*** Besides pattern occurrences, a workflow recipe also includes statistical characterizations of task performance metrics. These are necessary for generating representative workflow task instances (by sampling task runtime and input/output data sizes from appropriate probability distributions). Specifically, WfChef analyzes a set of real workflow instances for a particular application to produce statistical summaries of workflow performance characteristics, per task type. To this end, WfChef performs probability distributions fitting (minimizing the mean square error). Figure 3 shows an example of probability distribution fitting of task runtime for two task types from different workflow instances, by plotting the cumulative distribution function (CDF) of the data and the best probability distribution found. The outcome of this analysis applied to a set of workflow instances for a particular application is a summary that includes, for each task type, the best probability distribution fits for runtime, input data size, and output data size. For instance, Table 1 lists (for each workflow application for which WfInstances hosts instances) the probability distributions used for these fits. Listing 1 shows the statistical summary for one particular task type in the 1000Genome workflow application.

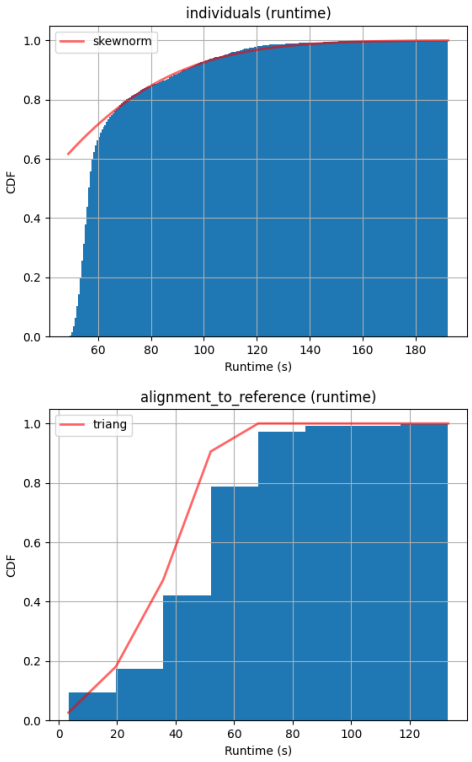



Figure 3: Example of probability distribution fitting of runtime (in seconds) for `individuals` tasks in 1000Genome workflows (*top*) and `alignment_to_reference` tasks in SoyKB (*bottom*) workflows.

### *3.3. WfGen*

Workflow instances are commonly used to drive experiments for evaluating novel workflow algorithms and systems. It is crucial to run large numbers of such experiments for many different workflow configurations, so as to ensure generality of obtained results. In addition, it is useful to conduct experiments while varying one or more characteristics of the workflow application, so as to study how these characteristics impact workflow execution. For instance, one may wish, for a particular overall workflow structure, to study how the workflow execution scales as the number of tasks increases. And yet, current archives only include instances for limited workflow configurations. And even as efforts are underway, including WfCommons, to increase the size of these archives, it is not realistic to expect them to include all relevant workflow configurations for all experimental endeavors. Instead, tools must be provided to generate representative *synthetic* workflow instances. These instances should be generated based on real workflow instances, so as to be representative, while conforming to user-specified characteristics, so as to be useful. The WfGen component targets the generation of such realistic synthetic workflow instances.

***Generating Synthetic Instances –*** WfGen takes as input a *workflow recipe* produced by WfChef for a particular application and a desired number of tasks. Note that each workflow recipe specifies a lower bound on the number of tasks that a generated synthetic workflow instance may contain. This is to ensure

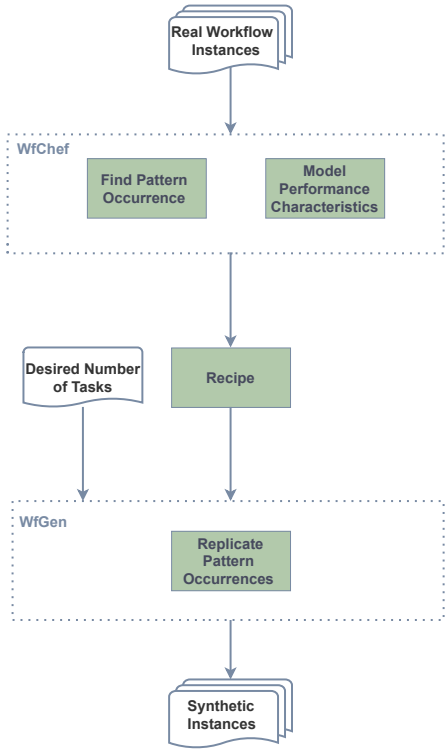



Figure 4: Overview of the synthetic workflow instance generation process. WfChef starts by analyzing a set of real workflow instances to discover pattern occurrences and compute statistical summaries of workflow task characteristics. It outputs a recipe that records this information. WfGen takes as input a recipe and a desired number of tasks, and replicates pattern occurrences in real-world instances to generate a synthetic instance with (approximately) the desired number of tasks.

that generated instances contain required application-specific structure. WfGen then automatically generates synthetic, yet realistic, randomized workflow instances with (approximately) the desired number of tasks. The way in which WfGen operates is depicted in the bottom part of Figure 4. To reach the designed number of tasks WfGen iteratively *replicates* pattern occurrences that are listed in the workflow recipes, randomly picking which pattern occurrence to replicate using a uniform probability distribution. Replicating a pattern occurrence simply consists in making a copy of all tasks and inter-task edges in the pattern occurrence to generate a new sub-graph. The entry and exit tasks of this sub-graph are connect to the same parent and children tasks as the entry and exit tasks of the original pattern occurrence. This process is repeated until replicating the next pattern occurrence would surpass the desired number of tasks.

### *3.4. WfSim*

An alternative to conducting scientific workflow research via real-world experiments is to use simulation. Simulation is used in many computer science domains and can address the limitations of real-world experiments. In particular, real-world experiments are confined to those application and platform configurations that are available to the researcher, and thus typically can only cover a small subset of the relevant scenarios that may be encountered in practice. Furthermore, real-world experiments can be time-, labor-, money-, and energy-intensive, as well as not perfectly reproducible.

WfCommons fosters the use of simulation for scientific workflow research, e.g., the development of workflow scheduling and resource provisioning algorithms, the development of workflow management systems, and the evaluation of current an emerging computing platforms for workflow executions. We do not develop simulators as part of the WfCommons project. Instead, WfCommons' WfSim component catalogs open source WMS simulators (such as those developed using the WRENCH framework [41, 42]) that support the WfFormat workflow instance format. In other words, these simulators take as input workflow instances in this format (either from actual workflow executions or synthetically generated) and simulate their executions. In the next section, we use two of the simulators cataloged in WfSim to quantify the extent to which synthetic instances generated using WfCommons tools are representative of real-world instances.

### *3.5. WfCommons Python package*

In order to allow users to analyze real workflow instances and to generate synthetic workflow instances, the WfCommons framework provides a collection of tools released as an open source Python package [19, 43]. Specifically, the package leverages the SciPy ecosystem [44] for performing probability distributions fitting to a series of data to produce statistical summaries of workflow performance characteristics (as described in Section 3.2). In contrast to our previous work [10], which used only two probability distributions for generating workflow performance metrics, the WfCommons Python package attempts to fit data with 23 probability distributions provided as part of SciPy's statistics submodule. Workflow recipes are instances of a class that defines methods for generating synthetic workflow instances given a desired number of tasks. The current version of the WfCommons Python package[3] provides recipes for generating synthetic workflows for all 9 applications shown in Table 1. Detailed documentation and examples can be found on the project's website [17] and the online open access package documentation [19].

## 4. Experimental Evaluation of Synthetic Generated Workflow Instances

In this section, we evaluate WfCommons and compare it to previously proposed approaches. We first evaluate the realism of generated workflow instances based on their structure (Section 4.2), and then based on their simulated execution using simulators of two state-of-the-art WMSs (Section 4.3).

### *4.1. Experimental Scenarios*

We consider experimental scenarios defined by particular workflow instances to be executed on particular platforms. To

[3] Version 0.7 was released in August 2021.

assess the realism and the accuracy of generated synthetic workflows, we have performed real workflow executions with Pegasus and Makeflow, and collected raw, time-stamped event logs from these executions. These logs form the ground truth to which we can compare simulated executions.

Actual workflow executions are conducted using the Chameleon Cloud platform [45], an academic cloud testbed on which we use homogeneous standard cloud units to run an HTCondor pool with shared file system, a submit node (which runs Pegasus or Makeflow), and a data node placed in the WAN. We use 4 worker "cloud units," where each each cloud unit consists of a 48-core 2.3GHz processor with 128 GiB of RAM. The bandwidth between the submit node and worker nodes on these instances is around 10Gbps. Simulated workflow executions are obtained based on the exact same hardware platform specification.

Whenever possible, for the experiments conducted in this section, we contrast experimental results obtained with synthetic workflow instances generated by *WfCommons* to results obtained using synthetic workflow instances generated by our previous work (*WorkflowHub*) [18]. For the *Epigenomics* and *Montage* applications we also include results obtained with an older, and very popular, framework called *WorkflowGenerator* [10]. We analyze a subset of 6 workflow applications, 4 from Pegasus and 2 from Makeflow (see Table 2). We chose these applications as they come from different science domains and have different graph structures and/or computing requirements.

### *4.2. Evaluating the Realism of Synthetic Workflow Instances*

To evaluate the structure of the generated synthetic workflow instances, we developed a metric called **Type Hash Frequency (THF)**. The THF metric is the Root Mean Square Error (RMSE) between the frequency of task type hashes for a synthetic workflow instance and that for the real workflow instance with the same number of tasks. Recall from Section 3.2 that a task's type hash encodes information about the task's type (kind of computation), but also about the task's ancestors and descendants. Therefore, the lower the THF of a synthetic workflow instance, the more similar it is to the real workflow instances.

For each real workflow instance of each selected application, as archived on WfCommons, we use WfCommons' Python package for generating a synthetic workflow instance with the same number of tasks. For comparison purposes, we also generate synthetic instances using the generators from our previous works.

Figure 5 shows THF results for all 6 applications, for WfCommons, WorkflowHub, and WorkflowGenerator. WorkflowGenerator only supports 2 of these application, which is why it is not included in all 6 plots. WfCommons and WorkflowHub use randomization in their heuristics, thus for each number of tasks we generate 10 sample synthetic workflows with each tool. The heights of the error bars in Figure 5 show average THF values, and error bars show the range between the third quartile (Q3) and the first quartile (Q1), in which 50% of the results lie. Error bars also show minimum and maximum values. Note that error bars are of zero length for the Blast (Figure 5e) and BWA (Figure 5f) applications. This is because workflows for these applications comprise a simple graph structure: there is only one task that can be replicated to produce synthetic workflow instances of different sizes. As a result, both WorkflowHub and WfCommons each produce ten identical synthetic workflow instances.

Overall, WfCommons yields the lowest THF values in most cases, often achieving low values in the absolute sense, meaning that the synthetic workflow instances it generates are representative of real workflow instances. Synthetic workflow instances produced by WorkflowGenerator have fixed graph structures, thus scaling up (resp. down) the number of tasks is simply done by replicating (resp. pruning) predefined subgraphs of the workflow. As a result, the generated workflow instances do not capture distinct patterns of the workflow graph produced by different sets of input data/parameters. For instance, for the Epigenomics workflow (Figure 5a) smaller instances are composed of a single or few chains of tasks, while larger instances are composed of several chains but also multiple branches (that can be composed of different numbers of chains). WorkflowGenerator is unable to capture this pattern. Synthetic workflow instances generated by the manually-crafted WorkflowHub generators are on average about 52% more realistic when compared to WorkflowGenerator. However, WorkflowHub still does not entirely capture all workflow patterns. For the Montage workflow (Figure 5b), real-world instances are obtained using two different image datasets (2MASS and DSS) [3]. Although these workflow instances are composed of the same set of executables, their graph structures differ significantly. WorkflowHub attempts to find a single structure to capture both cases, while WfCommons can precisely identify both distinct patterns. Similar results are observed for Cycles and 1000Genome (Figures 5c and 5d). For Blast and BWA (Figures 5e and 5f), THF values are very low, (but still with WfCommons leading to the best results) due to the simple structure of these workflows.

### *4.3. Evaluating the Accuracy of Synthetic Workflow Instances*

We use simulators of two state-of-the-art WMSs, Pegasus [3] and Makeflow [34], as a case study for evaluation and validation purposes. These simulators are described in [42] (note that the Makeflow simulator is really a simulator of WorkQueue, an execution engine used by Makeflow). Both Pegasus and Makeflow are being used in production to execute workflows for dozens of high-profile applications in a wide range of scientific domains and on a wide range of platforms. We used both systems to execute workflows on a cloud environment for the purpose of collecting execution logs for building real workflow instances, as described in Section 3.1. The simulators are built using WRENCH [41, 42], a framework for implementing simulators of WMSs that are accurate and can run scalably on a single computer, while requiring minimal software development effort. The work in [42] demonstrates that WRENCH achieves these objectives, and provides high simulation accuracy for workflow executions using both Pegagus and Makeflow.

In [42, 18], we have already demonstrated that the simulation framework used in our previous set of tools [10] suffers

Table 2: Applications, number of tasks and systems used on our experiments.

| Application | Number of Tasks per Workflow Instance | WMS |
|---|---|---|
| Blast | [45, 105, 305] | Makeflow |
| BWA | [106, 1006] | Makeflow |
| Cycles | [69, 135, 136, 203, 221, 268, 333, 401, 439, 440, 659, 663, 664, 876, 995, 1093, 1313, 1324, 1985, 2183, 2184, 3275, 4364, 6545] | Pegasus |
| Epigenomics | [43, 75, 121, 127, 225, 235, 243, 265, 349, 407, 423, 447, 509, 517, 561, 579, 673, 715, 795, 819, 865, 985, 1097, 1123, 1399, 1697] | Pegasus |
| 1000Genome | [54, 84, 106, 158, 166, 210, 248, 262, 314, 330, 366, 412, 418, 470, 494, 522, 574, 576, 658, 740, 822, 904] | Pegasus |
| Montage | [180, 312, 474, 621, 621, 750, 1068, 1314, 1740, 2124, 4848, 6450, 7119, 9807] | Pegasus |

from significant discrepancies from actual executions. These discrepancies mostly stem from the use of a simplistic network simulation model, and from the simulator not capturing relevant details of the system, and thus of the workflow execution. Therefore, to reach fair conclusions regarding the validity of synthetic workflow instances, in this paper we only use the more accurate WRENCH-based simulators for all experiments. Using these simulators, we quantify the extent to which the simulated execution of generated synthetic workflow instances (generated using our previous work and using WfCommons) is similar to that of real workflow instances. The simulator implementations, details on the calibration procedure, and experimental scenarios used in the rest of this section are all publicly available online [46, 47].

We perform experiments using simulators for the same subset of workflow applications as in the evaluation of the workflow generation. For each application, we run the simulator for a reference real workflow instance and for synthetic instances. The goal is to quantify the discrepancies between the simulated execution of a synthetic workflow instance and that of a real workflow instance with the same number of tasks, using the absolute relative difference between the simulated makespans (i.e., overall execution times in seconds). This metric is commonly used in the literature to quantify simulation error.

Figure 6 shows the relative error of simulated makespans for all 6 applications for WfCommons and WorkflowHub when compared to the real instances. The closest to zero the average value, the more realistic the synthetic instance. In this section, we omit all results for WorkflowGenerator as it performs very poorly, as is expected given the results in the previous section. Recall that WfCommons and WorkflowHub have randomization in their heuristics, therefore the values shown in Figure 6 are averages computed over 10 sample generated synthetic workflows for each tool and application.

Overall, we observe larger discrepancy between simulated executions of synthetic workflow instances and that of real workflow instances when the synthetic instances are generated by WorkflowHub. By contrast, WfCommons' generation method produces workflow instances whose executions are more closely matched to that of real workflow instances (even though some discrepancies necessarily remain due to random sampling effects). For Epigenomics (Figure 6a), both frameworks lead to many similar average makespan values, but WorkflowHub leads to significantly higher (and less realistic) makespans for instances comprised of 515, 577, 671, and 1397 tasks. By manually inspecting these generated synthetic instances, we observe a reduction in the number of branches in the workflow and an increase in the number of tasks in chains of tasks. As a result, the parallelism of the workflow is decreased, which explains the longer makespans. In the real instances, this parallelism reduction does not occur. WfCommons-generated instances are more representative of real instances but some discrepancies still occur for instances comprised of 713, 793, and 1695 tasks, albeit with smaller magnitude. Results for Montage (Figure 6b) corroborate the findings presented in Section 4.2, with a large advantage for WfCommons over WorkflowHub. Results for Cycles (Figure 6c) are similar, with WfCommons instances leading to makespans more in line with that of real instances for most cases. Simulated results for 1000Genome instances (Figure 6d) are similar to Epigenomics results, in that WorkflowHub leads to very inaccurate makespans for some instances. These large discrepancies occur for particular numbers of tasks (246, 328, 410, 492, 574, 656, and 738), which correspond to cases in which a new chromosome is added. WorkflowHub is not able to capture these changes in the workflow structures. By contrast, WfCommons leads to more accurate results, albeit with some remaining discrepancies. Finally, results for Blast (Figure 6e) and BWA (Figure 6f) show that both WorkflowHub and WfCommons lead to accurate makespans, which is expected given the simple structure of these workflows.

## 5. Case Study: Estimating Energy Consumption of Large-Scale Workflows

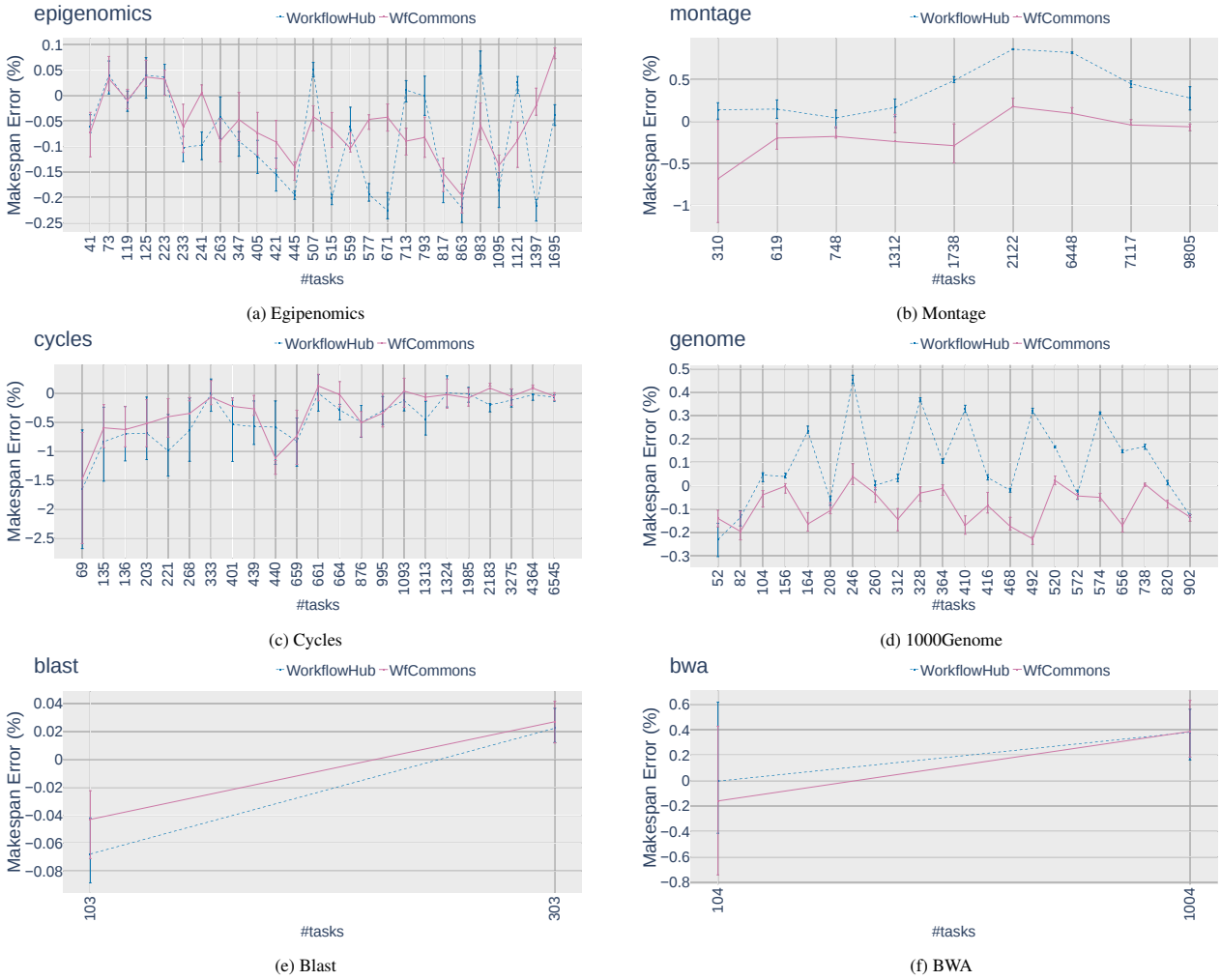
Energy-efficient computing has received much attention in the past few years. With the advance of computing capabilities, applications become more complex and consume more resources, thus leading to increased energy usage [48]. While the development of fully renewable computing facilities is on the rise, there is still a pressing need to reduce the power consumption of computation, which in turn reduces its carbon footprint. In the context of scientific workflows, several works have proposed solutions to optimize workflow executions while respecting energy consumption constraints [49].

In [50, 51], we have proposed and validated a power consumption model that accounts for CPU utilization, computations that execute on multi-core compute nodes, and I/O operations (including the idle power consumption caused by waiting for these operations to complete). In this work, we leverage this model, to estimate the energy consumption of the execution of large-scale workflows. We use Montage workflows as a case-study and compute the estimated energy consumption of the execution of real workflow instances using our model. We then compute the estimated energy consumption of the execution of synthetic Montage instances generated by WfCommons. We generate synthetic instances with approximately the same


Figure 5: THF of synthetic workflow instances. Bar heights are average values. Error bars show the range between the third quartile (Q3) and the first quartile (Q1), and minimum and maximum values as black dots.

number of tasks as their real counterparts, so as to validate the accuracy of the generated instances. Additionally, we generate larger instances of that available real Montage workflow instances, that is, with 10K, 25K, 50K, 75K, 100K, 150K, 200K, and 250K tasks. The goal is to demonstrate the usefulness of WfCommons in assessing the energy consumption of Montage applications at scales for which data from actual executions is not available.

Figure 7 shows simulated energy consumption vs. number of workflow tasks, for both real and synthetic Montage work-




(a) Egipenomics

(b) Montage

(c) Cycles

(d) 1000Genome

(e) Blast

(f) BWA

Figure 6: Relative error of simulated makespan for workflow instances. Square markers show average values. Error bars show the range between the third quartile (Q3) and the first quartile (Q1).

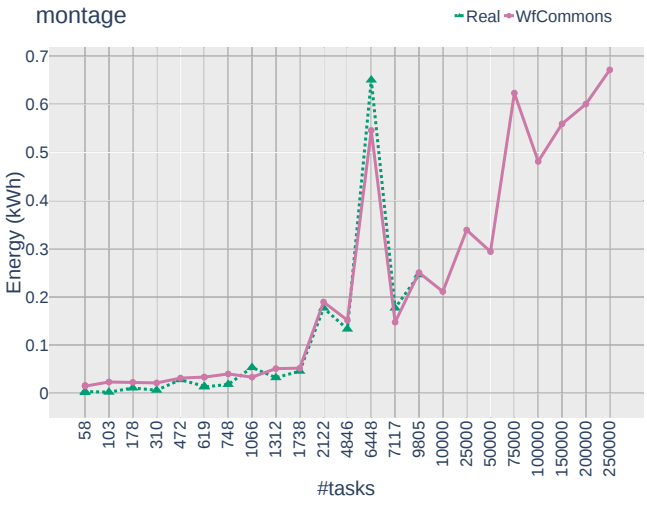



Figure 7: Estimated energy consumption (in kWh) of the (simulated) executions of synthetic and real Montage workflow instances. Synthetics instances are generated at scales beyond that of the available real instances.

flow instances. Simulated executions are for the same hardware platform specification as that described in Section 4.1. Intriguingly, energy consumption for real instances does not monotonically increase with the number of workflow tasks. Similar non-monotonic behavior is observed for all other 5 applications considered in this work (results not shown). For Montage instances that comprise 2,212 and 6,448 tasks we note large energy consumption spikes. Manual inspection of these instances reveals that the number of tasks in the fan-out portions of the workflow graph significantly diminishes when compared to the adjacent instances (which have fewer branches and stretched fan-out patterns). This application-specific feature leads to a reduction of the number of tasks that are ready for execution at particular points in time, thus causes the WMS (or rather the scheduling algorithm it employs) to underutilize the resources. This lowered resource utilization results in higher makespans (as seen in Figure 6b). It also results in a large increase in energy consumption due to the inherent energy consumption of having machines in operation, even if idle (i.e., due to *static power consumption*).

The key results here is that synthetic workflow instances

generated by WfCommons produce very similar energy consumption profiles to that of the result instances. More importantly, the spikes observed for the real instances are also observed for synthetic instances. This shows that WfCommons is able to account for application-specific features and patterns accurately when producing workflow recipes. Figure 7 shows 8 data points for synthetic instances that go beyond the scale of available real instances. These data points show a large energy-consumption spike for the instance with 75K tasks. Given the accuracy of the energy consumption estimate with synthetic instances for scales up to 9,805 tasks, there is good confidence that a real Montage execution with 75K tasks would also experience a large energy consumption spike.

The overall conclusion from the above results, which corroborates results in previous sections, is that WfCommons generates accurate (both at structural and performance metrics level) synthetic workflow instances. These instances can be used to study workflow execution behavior, as demonstrated for energy consumption behavior in this case study, at scales beyond that for which real execution data is available.

## 6. Conclusion

In this paper, we have presented the WfCommons project, a community framework for constructing and archiving workflow instances, analyzing these instances, producing realistic synthetic workflow instances, and simulating workflow executions using these instances. WfCommons provides a collection of tools for constructing "workflow recipes" based on instances collected from the real-world execution of workflow applications. These workflow recipes can then be used to produce synthetic workflow instances. These synthetic instances that can enable a variety of novel workflow systems research and development activities. We have demonstrated experimentally that the synthetic instances generated by WfCommons are realistic, much more so than those produced by previously available generators. More specifically, WfCommons is able to generate synthetic workflow instances at various scales while preserving key application-specific structural patterns and performance characteristics. We have showcased the usefulness of WfCommons via a case study focused on the energy consumption of workflow executions. Specifically, we have shown that using the generated synthetic workflow instances lead to experimental results that are in line with that obtained with real workflow instances, while making it possible to explore scenarios for workflow scales beyond that of available real workflow instances.

WfCommons is open-source and welcomes contributors. It currently provides a collection of 165 workflow instances derived from actual executions, and can generate synthetic workflows from 9 applications from 4 science domains. Version 0.7 was released in August 2021. We refer the reader to `https://wfcommons.org` for software, documentation, and links to collections of instances and simulators.

A short-term future work direction is the development of additional execution logs parsers for state-of-the-art workflow systems [52]. These parsers will enable WfCommons to provide continuous automated development of novel workflow recipes to broaden the number of science domains in which WfCommons can potentially impact research and development efforts. Another future work direction is the use of synthetic workflow instances to support workflow-focused education and training, e.g., for designing simulation-driven activities in which students acquire knowledge by experimenting with various workflow scenarios [53].

## Acknowledgments

This work is funded by NSF contracts #2016610, and #2016619; and partly funded by DOE contract number #DE-SC0012636, and NSF contracts #2103489, #2103508, #1923539, #1923621, and #1664162. We also thank the NSF Chameleon Cloud for providing time grants to access their resources.

## References

[1] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. Ferreira da Silva, G. Papadimitriou, M. Livny, The Evolution of the Pegasus Workflow Management Software, Computing in Science & Engineering 21 (4) (2019) 22–36. doi:10.1109/MCSE.2019.2919690.

[2] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, J. I. V. Hemert, Scientific Workflows: Moving Across Paradigms, ACM Computing Surveys (CSUR) 49 (4) (2016) 1–39. doi:10.1145/3012429.

[3] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, K. Wenger, Pegasus, a workflow management system for science automation, Future Generation Computer Systems 46 (0) (2015) 17–35. doi:10.1016/j.future.2014.10.008.

[4] A. Klimentov, P. Buncic, K. De, S. Jha, T. Maeno, R. Mount, P. Nilsson, D. Oleynik, S. Panitkin, A. Petrosyan, R. J. Porter, K. F. Read, A. Vaniachine, J. C. Wells, T. Wenaus, Next Generation Workload Management System For Big Data on Heterogeneous Distributed Computing, Journal of Physics: Conference Series 608 (1) (2015) 12040. doi:10.1088/1742-6596/608/1/012040.

[5] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, J. Vetter, The future of scientific workflows, International Journal of High Performance Computing Applications 32 (1) (4 2017). doi:10.1177/1094342017704893.

[6] R. Ferreira da Silva, H. Casanova, K. Chard, D. Laney, D. Ahn, S. Jha, C. Goble, L. Ramakrishnan, L. eerson, B. Enders, D. Thain, I. Altintas, Y. Babuji, R. Badia, V. Bonazzi, T. Coleman, M. Crusoe, E. Deelman, F. Di Natale, P. Di Tommaso, T. Fahringer, R. Filgueira, G. Fursin, A. Ganose, B. Gruning, D. S. Katz, O. Kuchar, A. Kupresanin, B. Ludascher, K. Maheshwari, M. Mattoso, K. Mehta, T. Munson, J. Ozik, T. Peterka, L. Pottier, T. Randles, S. Soiland-Reyes, B. Tovar, M. Turilli, T. Uram, K. Vahi, M. Wilde, M. Wolf, J. Wozniak, Workflows Community Summit: Bringing the Scientific Workflows Community Together (Mar. 2021). doi:10.5281/zenodo.4606958.

[7] L.-C. Canon, A. K. W. Chang, Y. Robert, F. Vivien, Scheduling independent stochastic tasks under deadline and budget constraints, The International Journal of High Performance Computing Applications 34 (2) (2020) 246–264. doi:10.1177/1094342019852135.

[8] L. Han, V. L. Fèvre, L.-C. Canon, Y. Robert, F. Vivien, A generic approach to scheduling and checkpointing workflows, The International Journal of High Performance Computing Applications 33 (6) (2019) 1255–1274. doi:10.1177/1094342019866891.

[9] T. Coleman, H. Casanova, T. Gwartney, R. Ferreira da Silva, Evaluating energy-aware scheduling algorithms for I/O-intensive scientific workflows, in: International Conference on Computational Science (ICCS), 2021. doi:10.1007/978-3-030-77961-0_16.

[10] R. Ferreira da Silva, W. Chen, G. Juve, K. Vahi, E. Deelman, Community Resources for Enabling and Evaluating Research in Distributed Scientific Workflows, in: 10th IEEE International Conference on e-Science, eScience'14, 2014, pp. 177–184. doi:10.1109/eScience.2014.44.

[11] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, L. Liu, Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds, IEEE Transactions on Parallel and Distributed Systems 27 (12) (2016) 3501–3517. doi:10.1109/TPDS.2016.2543731.

[12] M. A. Rodriguez, R. Buyya, Scheduling dynamic workloads in multitenant scientific workflow as a service platforms, Future Generation Computer Systems 79 (2018) 739–750. doi:10.1016/j.future.2017.05.009.

[13] W. Chen, R. Ferreira da Silva, E. Deelman, R. Sakellariou, Using imbalance metrics to optimize task clustering in scientific workflow executions, Future Generation Computer Systems 46 (2015) 69–84. doi:10.1016/j.future.2014.09.014.

[14] Z. Tong, H. Chen, X. Deng, K. Li, K. Li, A scheduling scheme in the cloud computing environment using deep q-learning, Information Sciences 512 (2020) 1170–1191. doi:10.1016/j.ins.2019.10.035.

[15] T. A. Genez, L. F. Bittencourt, N. L. da Fonseca, E. R. Madeira, Estimation of the available bandwidth in inter-cloud links for task scheduling in hybrid clouds, IEEE Transactions on Cloud Computing 7 (1) (2015) 62–74. doi:10.1109/TCC.2015.2469650.

[16] R. Ferreira da Silva, H. Casanova, R. Tanaka, F. Suter, Bridging concepts and practice in escience via simulation-driven engineering, in: Workshop on Bridging from Concepts to Data and Computation for eScience (BC2DC'19), 15th International Conference on eScience (eScience), 2019, pp. 609–614. doi:10.1109/eScience.2019.00084.

[17] WfCommons Project, https://wfcommons.org (2021).

[18] R. Ferreira da Silva, L. Pottier, T. Coleman, E. Deelman, H. Casanova, WorkflowHub: Community Framework for Enabling Scientific Workflow Research and Development, in: 2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS), IEEE, 2020, pp. 49–56. doi:10.1109/WORKS51914.2020.00012.

[19] WfCommons Python package, https://docs.wfcommons.org (2021).

[20] D. G. Feitelson, D. Tsafrir, D. Krakov, Experience with using the Parallel Workloads Archive, Journal of Parallel and Distributed Computing 74 (10) (2014) 2967–2982. doi:10.1016/j.jpdc.2014.06.013.

[21] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D. H. Epema, The Grid Workloads Archive, Future Generation Computer Systems 24 (7) (2008) 672–686. doi:10.1016/j.future.2008.02.003.

[22] D. Kondo, B. Javadi, A. Iosup, D. Epema, The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems, in: 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing, IEEE, 2010, pp. 398–407. doi:10.1109/CCGRID.2010.71.

[23] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, M. Scales, S. Soiland-Reyes, L. Stojanovic, Common workflow language, v1.0, figshare (2016).

[24] L. Versluis, R. Mathá, S. Talluri, T. Hegeman, R. Prodan, E. Deelman, A. Iosup, The Workflow Trace Archive: Open-Access Data From Public and Private Computing Infrastructures, IEEE Transactions on Parallel and Distributed Systems 31 (9) (2020) 2170–2184. doi:10.1109/TPDS.2020.2984821.

[25] M. A. Amer, R. Lucas, Evaluating Workflow Tools with SDAG, in: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, IEEE, 2012, pp. 54–63. doi:10.1109/SC.Companion.2012.20.

[26] DAGGEN: A synthetic task graph generator, https://github.com/frs69wq/daggen (2021).

[27] D. G. Amalarethinam, G. J. Mary, DAGEN - A Tool To Generate Arbitrary Directed Acyclic Graphs Used For Multiprocessor Scheduling, International Journal of Research and Reviews in Computer Science 2 (3) (2011) 782. doi:notavailable.

[28] D. G. Amalarethinam, P. Muthulakshmi, DAGITIZER – A Tool to Generate Directed Acyclic Graph through Randomizer to Model Scheduling in Grid Computing, in: Advances in Computer Science, Engineering & Applications, Springer, 2012, pp. 969–978. doi:10.1007/978-3-642-30111-7_93.

[29] D. Garijo, P. Alper, K. Belhajjame, O. Corcho, Y. Gil, C. Goble, Common motifs in scientific workflows: An empirical analysis, Future Generation Computer Systems 36 (2014) 338–351. doi:10.1016/j.future.2013.09.018.

[30] W. M. van Der Aalst, A. H. Ter Hofstede, B. Kiepuszewski, A. P. Barros, Workflow Patterns, Distributed and Parallel Databases 14 (1) (2003) 5–51. doi:10.1023/A:1022883727209.

[31] U. Yildiz, A. Guabtni, A. H. Ngu, Towards scientific workflow patterns, in: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, 2009, pp. 1–10.

[32] D. S. Katz, A. Merzky, Z. Zhang, S. Jha, Application skeletons: Construction and use in eScience, Future Generation Computer Systems 59 (2016) 114–124. doi:10.1016/j.future.2015.10.001.

[33] WfCommons JSON Schema, https://github.com/wfcommons/workflow-schema (2021).

[34] M. Albrecht, P. Donnelly, P. Bui, 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, 2012, pp. 1–13. doi:10.1145/2443416.2443417.

[35] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future generation computer systems 29 (3) (2013) 682–692. doi:10.1016/j.future.2012.08.015.

[36] R. Ferreira da Silva, R. Filgueira, E. Deelman, E. Pairo-Castineira, I. M. Overton, M. P. Atkinson, Using simple pid-inspired controllers for online resilient resource management of distributed scientific workflows, Future Generation Computer Systems 95 (2019) 615–628. doi:10.1016/j.future.2019.01.015.

[37] C. Zheng, D. Thain, Integrating containers into workflows: A case study using makeflow, work queue, and docker, in: Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing, 2015, pp. 31–38. doi:10.1145/2755979.2755984.

[38] R. Ferreira da Silva, R. Mayani, Y. Shi, A. R. Kemanian, M. Rynge, E. Deelman, Empowering agroecosystem modeling with htc scientific workflows: The cycles model use case, in: First International Workshop on Big Data Tools, Methods, and Use Cases for Innovative Scientific Discovery (BTSD), 2019, pp. 4545–4552. doi:10.1109/BigData47090.2019.9006107.

[39] R. Ferreira da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, E. Deelman, A characterization of workflow management systems for extreme-scale applications, Future Generation Computer Systems 75 (2017) 228–238. doi:10.1016/j.future.2017.02.026.

[40] T. Coleman, H. Casanova, R. F. da Silva, WfChef: Automated Generation of Accurate Scientific Workflow Generators, arXiv preprint arXiv:2105.00129 (2021).

[41] H. Casanova, S. Pandey, J. Oeth, R. Tanaka, F. Suter, R. Ferreira da Silva, WRENCH: A Framework for Simulating Workflow Management Systems, in: 13th Workshop on Workflows in Support of Large-Scale Science (WORKS'18), 2018, pp. 74–85. doi:10.1109/WORKS.2018.00013.

[42] H. Casanova, R. Ferreira da Silva, R. Tanaka, S. Pandey, G. Jethwani, W. Koch, S. Albrecht, J. Oeth, F. Suter, Developing Accurate and Scalable Simulators of Production Workflow Management Systems with WRENCH, Future Generation Computer Systems 112 (2020) 162–175. doi:10.1016/j.future.2020.05.030.

[43] WfCommons GitHub Repository, https://github.com/wfcommons/wfcommons (2021).

[44] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al., SciPy 1.0: fundamental algorithms for scientific computing in Python, Nature methods 17 (3) (2020) 261–272. doi:10.1038/s41592-019-0686-2.

[45] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, et al., Lessons Learned from the Chameleon Testbed, in: 2020 USENIX Annual Technical Conference, 2020, pp. 219–233. doi:notavailable.

[46] WRENCH Pegasus Simulator, https://github.com/wrench-project/pegasus (2021).

[47] WRENCH WorkQueue Simulator, https://github.com/wrench-project/workqueue (2021).

[48] M. Zakarya, Energy, performance and cost efficient datacenters: A survey, Renewable and Sustainable Energy Reviews 94 (2018) 363–385. doi:10.1016/j.rser.2018.06.005.

[49] A.-C. Orgerie, M. D. d. Assuncao, L. Lefevre, A survey on techniques for improving the energy efficiency of large-scale distributed systems, ACM Computing Surveys (CSUR) 46 (4) (2014) 1–31. doi:10.1145/2532637.

[50] R. Ferreira da Silva, A.-C. Orgerie, H. Casanova, R. Tanaka, E. Deelman, F. Suter, Accurately Simulating Energy Consumption of I/O-intensive Scientific Workflows, in: Computational Science – ICCS 2019, Springer International Publishing, 2019, pp. 138–152. doi:10.1007/978-3-030-22734-0_11.

[51] R. Ferreira da Silva, H. Casanova, A.-C. Orgerie, R. Tanaka, E. Deelman, F. Suter, Characterizing, Modeling, and Accurately Simulating Power and Energy Consumption of I/O-intensive Scientific Workflows, Journal of Computational Science 44 (2020) 101157. doi:10.1016/j.jocs.2020.101157.

[52] Existing workflow systems, https://s.apache.org/existing-workflow-systems (2021).

[53] R. Tanaka, R. Ferreira da Silva, H. Casanova, Teaching Parallel and Distributed Computing Concepts in Simulation with WRENCH, in: Workshop on Education for High-Performance Computing (EduHPC), 2019, pp. 1–9. doi:10.1109/EduHPC49559.2019.00006.