

Selecting Scheduling Algorithms at Runtime using Simulation

Henri Casanova¹[0000-0001-6310-0365],
Yick Ching Wong¹[0000-0002-3837-461X],
Loïc Pottier²[0000-0002-7681-3521], and
Rafael Ferreira da Silva³[0000-0002-1720-0928]

¹ Information and Computer Sciences Department
University of Hawaii, Honolulu, HI, USA
[henric,wongy]@hawaii.edu

² Information Sciences Institute
University of Southern California, Marina Del Rey, CA, USA
lpottier@isi.edu

³ National Center for Computational Sciences
Oak Ridge National Laboratory, Oak Ridge, TN, USA**
silvarf@ornl.gov

Abstract. Runtime systems that automate the execution of applications on distributed cyberinfrastructures need to make scheduling decisions. Researchers have proposed many scheduling algorithms, but most of them are designed based on analytical models and assumptions that may not hold in practice. The literature is thus rife with algorithms that have been evaluated only within the scope of their underlying assumptions but whose practical effectiveness is unclear. It is thus difficult for developers to decide which algorithm to implement in their runtime systems.

To obviate the above difficulty, we propose an approach by which the runtime system executes, throughout application execution, simulations of this very execution. Each simulation is for a different scheduling algorithm, and the best algorithm is selected based on simulation results. These simulations do not make any of the simplified assumptions that may have been used to develop the algorithms, and thus allow for realistic algorithm comparisons. But simulations are never perfectly accurate. The main objective of this work is to evaluate the feasibility and potential merit of this approach, even in the presence of simulation inaccuracy, when compared to the traditional one-algorithm approach. We perform this evaluation via a case study in the context of scientific workflows. Our main finding is that our proposed approach can outperform the best one-algorithm approach even in the presence of relatively large simulation inaccuracies.

Keywords: Scheduling · On-line Simulation · Workflows

** This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The publisher acknowledges the US government license to provide public access under the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

1 Introduction

Data processing and analysis applications that execute on parallel and distributed computing environments, or CyberInfrastructures (CI), arise in most fields of science and engineering. A key endeavor has been to develop CI runtime systems that make it straightforward for users to implement, deploy, and execute their applications. To this end, all these systems automate application execution, including the resource management and task scheduling decision making process. Specifically, decisions must be made along, at least, the following axes:

- Selecting hardware and/or virtualized resources;
- Picking application configuration options (e.g., pick numbers of cores that should be used by multi-threaded tasks);
- Scheduling application activities in time (when?) and space (which resource?).

Decisions along these axes must be made so as to meet user-level objectives and constraints, which can encompass notions of performance, monetary cost, energy consumption, reliability, etc. For simplicity, we call all above decisions *scheduling decisions*, which must be made using *scheduling algorithms*. Scheduling problems are generally NP-complete, and thus most proposed algorithms employ non-guaranteed heuristics for exploring a large decision space.

The design of scheduling algorithms has received an enormous amount of effort. For instance, solely in the context of the popular “scientific workflow” application model [4], hundreds of research publications propose scheduling algorithms (see the many surveys on this topic [1,3,13,17,19,24,28]). Most of these proposed algorithms reuse ideas and principles from the age-old and extensive DAG (Directed Acyclic Graph) scheduling literature [29]. Yet, when examining existing workflow runtime systems, there is a clear disconnect between research and practice. Given the complexity of CI platforms and applications, research results are typically obtained based on simplifying analytical models and assumptions, so that scheduling problems are rendered more formalizable and tractable. For instance, ignoring network contention greatly simplifies application scheduling problems [12], but the computed schedules will perform poorly in practice when network contention does occur. Furthermore, published evaluation results for proposed algorithms cannot cover the whole range of situations a runtime system could encounter in practice. The literature is thus rife with scheduling algorithms that have been evaluated within the scope of their underlying assumptions, but whose potential effectiveness in practice is unquantified.

Given the above, there is little incentive for developers of CI runtime systems to pay close attention to scheduling research. Based on our own observation of production systems, it seems that developers often opt for simple scheduling strategies that are straightforward to implement. As a result, scheduling decisions made by state-of-the-art runtime systems suffer from two limitations: (i) Relevant scheduling objectives cannot be achieved by these simple strategies, such as multi-objective, multi-constraints objectives (e.g., achieve user-defined compromises between execution time and energy consumption, minimize energy consumption while completing the application by a deadline); (ii) Even when targeting a single objective, such as minimizing execution time, the simple

scheduling strategies implemented in state-of-the-art CI systems lead to sometimes quantified [25,27], but typically unquantified, lost opportunities for more efficient application executions.

We seek to resolve the above disconnect between scheduling research and practice by obviating the challenge of picking one particular scheduling algorithm to implement as part of a CI runtime system. To this end, we propose the use of *online simulations* for picking which algorithm to use at runtime. In other words, the key idea is to execute fast simulations of the application execution throughout that very execution so as to “try out” many potential scheduling algorithms and automatically select the most desirable one. A CI runtime system can then implement several scheduling algorithms, and use simulation to dynamically pick which one to use at various points during application execution. As discussed above, many of these algorithms operate with models and assumptions that are not realistic, which makes their effectiveness unclear in practice. The simulation, however, implements realistic models and assumptions, which will reveal how scheduling algorithms would actually perform in practice. As a result, some of the implemented algorithms may rarely (or even never) be used at runtime because simulations show them to be non-competitive. CI runtime system developers can incrementally add to the quiver of implemented algorithms in their system, without ever having to decide at compile time which algorithm should be used.

Our objective in this work is to assess the feasibility and potential merit of an online simulation approach for driving scheduling decisions in CI runtime systems. Although our approach is general, we perform our experimental evaluations in the specific context of scientific workflows because they have become widespread as well as the CI runtime systems available to execute them. More specifically, this work makes the following contributions:

- We propose an online simulation approach for transforming the way in which CI runtime systems that automate the execution of application workloads perform their scheduling decisions;
- We evaluate the feasibility and potential merit of this approach via a case study to answer three main research questions: (i) What is the potential improvement over the traditional one-algorithm approach? (ii) How much of the upcoming application execution should be simulated? (iii) What level of simulation accuracy is needed?
- Our main finding is that our proposed approach outperforms the best one-algorithm approach even in the presence of simulation inaccuracies.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 describes our approach, which we evaluate via the case study described in Section 4. Section 5 discusses experimental results. Finally, Section 6 summarizes our contributions and highlights directions for future work.

2 Related Work

There is an enormous literature devoted to proposing models, algorithms, and practical approaches for computing solutions to scheduling problems. As ex-

plained in Section 1, there is a significant gap between theory and practice. This gap is seen, for instance, in the area of workflow scheduling, which is itself the object of an extensive literature [1, 3, 13, 17, 19, 24, 28] and, which we target in this work as a case study. We attempt to bridge this gap not by proposing yet another algorithm but by using online simulation to select which algorithm to use at runtime throughout application execution.

To the best of our knowledge, in the field of parallel and distributed computing, the idea of using simulations to drive scheduling decisions at runtime has not received much attention. In [7], the authors propose to use online simulation to improve the decisions made by an HPC batch scheduler. The approach consists in simulating several possible execution orders of jobs in the batch queue, computing a score based on how each ordering would affect average bounded slowdown (which is to be minimized). A scheduling policy is then defined at runtime based on score distributions. It is not clear how that approach can be applied to contexts in which the scheduling problem does not boil down to merely ordering jobs or tasks. In this work, because we want to target general scheduling problems, we instead use online simulation to compare different, previously proposed scheduling algorithms. In [20], the authors note that the simplified analytical models with which scheduling algorithms are designed negatively impact the quality of scheduling decisions in practice. They focus on a list-scheduling algorithm that uses a simplistic model of data transfer times. They propose to augment this algorithm so that, at runtime, it uses online simulation to compute more accurate network transfer time estimates. The main drawback of their approach is the high simulation overhead because each scheduling decision requires the execution of online simulations. In this work, instead, online simulations are only executed occasionally throughout application execution. Another key difference is that our approach is not confined to using a single algorithm, but instead uses online simulation to evaluate a potentially large number of different algorithms.

Although the idea of driving scheduling using online simulations has been only rarely explored in the context of parallel and distributed computing, it has received attention in other contexts, such as for manufacturing and logistics (e.g., to solve versions of the job shop scheduling problem for cyber-physical systems). A cursory review of that literature shows that simulation-driven scheduling has been proposed and used since at least the early 1990’s. To take a very recent example, the work in [22] proposes simulation-driven scheduling in the context of wind farms operation.

Many simulation frameworks have been developed that target the simulation of parallel and distributed applications and platforms [5, 6, 8–10, 14, 15, 18, 21, 23, 30], and they each achieve different compromises between accuracy and speed. At one extreme are discrete-event models that capture “microscopic” behaviors of hardware/software systems (e.g., packet-level network simulation, block-level disk simulation, cycle-accurate CPU simulation), which favor accuracy over speed. At the other extreme are analytical models that capture “macroscopic” behaviors via mathematical models. While these models lead to fast simulation,

they must be developed carefully if high levels of accuracy are to be achieved [31]. This work is agnostic to the simulation framework used to implement the simulation, but a more accurate and more scalable framework is obviously preferable. For the case study in Section 4, we implement a simulator using the SimGrid [9] and WRENCH [10] frameworks. SimGrid provides accurate and scalable simulation models and abstractions for simulating distributed applications, systems, and platforms. To date, it has been used to obtain simulation results for 550+ research publications. One drawback of SimGrid is that its simulation abstractions are low-level, meaning that implementing simulators of complex systems can be labor-intensive [16]. WRENCH builds on SimGrid to provide high-level simulation abstractions that make it possible to implement simulators of complex CI scenarios in only a few hundred lines of code [10].

3 Problem Statement, Approach, Research Questions

Consider a CI platform with hardware resources (compute, storage, network) accessible via various software services for starting computations, storing data, and moving data. Some application workload of interest is to be executed on this platform. A CI runtime system is used to automate this execution, and as part of this automation the system must make decisions regarding the allocation of application activities to the hardware resources in time and space. These scheduling decisions are made using some algorithm, with the goal of optimizing some metric such as overall execution time.

We propose to use simulation to search the scheduling decision space at runtime. The main caveat of scheduling algorithms in the literature is that they are developed with simplifying models and assumptions so as to make the scheduling problem algorithmically more tractable. By contrast, simulation does not need to make simplifying assumptions. For instance, it can easily capture stochastic platform and application behaviors, complex network sharing behaviors, or complex overlap behaviors between computation, I/O, and network communication activities. Although accounting for such behaviors makes the scheduling problem algorithmically more difficult, our approach does not entail solving this problem algorithmically at all. Instead, simulations merely output relevant application-level metrics (e.g., execution time, cost, energy consumption, reliability) for many candidate scheduling algorithms, and one can simply pick the most desirable one. In a nutshell, we shift the scheduling problem from “come up with one algorithm for solving all instances of a difficult multi-objective, multi-constraint scheduling problem” to “enumerate a set of solutions computed by several algorithms for the particular problem instance at hand, and pick the best one based on simulated executions”. These algorithms must be implemented in the runtime system. At the onset of application execution, a description of the application and the available hardware resources is constructed based on (likely imperfect) a-priori knowledge, so as to instantiate a simulator of the upcoming application execution. Throughout execution, scheduling decisions are made using one of the implemented algorithms, selected based on simulation results.

Realizing the above approach in practice entails addressing many research and engineering challenges that are outside the scope of this work. Our objective here is to determine whether this approach has potential merit in the first place. To this end, we focus on the following research questions:

How much of an improvement can the online simulation approach afford? We wish to compare our proposed approach to the traditional one-algorithm approach in which the runtime system uses a single scheduling algorithm throughout application execution. Assuming that a significant improvement is achieved, intriguing questions arise regarding the usefulness of individual algorithms (i.e., how many algorithms are never used? how many different algorithms are used throughout application execution?).

How much of the upcoming application execution should be simulated? In spite of advances in scalable simulation techniques for simulating distributed applications and platforms, online simulations do not take zero time. One easy way to reduce simulation overhead is to bound the simulated time horizon and not simulate the upcoming application execution until completion. We wish to quantify the impact of making simulations “short-sighted” on the effectiveness of our proposed approach.

What level of simulation accuracy is needed? Simulations are never 100% accurate, because of inaccuracies inherent to the simulation models or because model parameters are not instantiated in a way that perfectly matches real-world settings. We wish to determine what level of simulation accuracy is needed for our proposed approach to outperform the traditional one-algorithm approach.

We answer these questions via the case study described in the next section.

4 Case Study

We consider the execution of scientific workflow applications on a multi-cluster CI deployment, where the goal is to minimize overall execution time, or *makespan*. Scientific workflows have been used by computational scientists to support some of the most significant discoveries of the past several decades [4], and are executed daily to serve a wealth of scientific domains. Many workflows have high computational demands and, as such, are executed in production on HPC clusters. Setting up, orchestrating, monitoring, and optimizing workflow executions on these platforms is challenging, and the way to address this challenge is to rely on runtime systems, or Workflow Management Systems (WMSs), that can automate workflow execution [26]. The past decade has witnessed a proliferation of WMSs [32]. Many scheduling algorithms that could be implemented in WMSs have been proposed in the literature [1, 3, 13, 17, 19, 24, 28], often borrowing ideas from the traditional task graph scheduling [29]. Yet, there is no consensus on which algorithms are best, all proposed algorithms make simplifying assumptions that make their practical efficacy unclear, and most proposed algorithms are never used in practice. It is for these reasons that we picked this context for this case study.

Table 1: Multi-cluster platform configurations used for experiments. Each cluster is defined by a number of nodes (“nodes”), a core speed in Gflop/sec (“speed”), and an Internet bandwidth in MBps (“bdwidth”).

Config	Cluster #1			Cluster #2			Cluster #3		
	nodes	speed	bdwidth	nodes	speed	bdwidth	nodes	speed	bdwidth
P_1	96	100	100		n/a			n/a	
P_2	48	50	100	48	150	100		n/a	
P_3	48	50	100	48	400	10		n/a	
P_4	32	100	100	32	200	200	32	300	300
P_5	32	100	100	32	200	300	32	300	200
P_6	32	100	200	32	200	100	32	300	300
P_7	32	100	200	32	200	300	32	300	100
P_8	32	100	300	32	200	200	32	300	100
P_9	32	100	300	32	200	100	32	300	200

4.1 Platform configurations

We consider multi-cluster platforms. Each cluster hosts homogeneous 8-core compute nodes connected via a 100GbE interconnect, as well as network-attached storage with I/O read/write bandwidths of 100MBps. Each cluster is connected to the Internet on a network path with some bottleneck bandwidth. The network-attached storage is used to cache application data. That is, whenever a compute node in a cluster needs to write application data, it writes it to the cluster’s network-attached. Whenever a compute node in a cluster needs to read application data, it does so from the network-attached storage if possible. Otherwise, the data is read from a remote location (the user’s machine, where all input data is located initially, or another cluster’s network-attached storage) and cached locally. We assume that storage capacity at each cluster is large enough to hold all application data.

We conduct experiments with the 9 synthetic 1-, 2-, and 3-cluster platform configurations listed in Table 1. These configurations do not correspond to particular real-world platforms and many other configurations could be considered. Our goal is to include configurations for which different scheduling algorithms would make different decisions (e.g., due to the different ratios of compute speed to Internet bandwidth for the clusters in configurations P4 to P9).

4.2 Workflow configurations

We consider 8 real-world scientific workflow instances, as listed in Table 2. These instances are provided by the WfCommons project⁴ and were derived based on logs from actual executions [11]. Each instance defines a set of tasks, each with particular amounts of computation to perform, and input and output files of particular sizes. Some output files of a task are input files to other tasks, thus creating data dependencies between tasks. We selected instances whose work

⁴ <https://wfcommons.org/instances>

Table 2: Workflow configurations used in our experiments, indicating for each the application name (“name”), the application domain (“domain”), the number of tasks (“tasks”), the sequential compute time in hours on a single 100Gflop/sec core (“work”), the sum of all data file sizes (“footprint”), the number of levels (“depth”), and the size of the largest level (“max width”).

Config	name	domain	tasks	work	footprint	depth	max width
W_1	Montage	Astronomy	4,846	8.7	12.15 GB	8	3,411
W_2	Epigenomics	Bioinformatics	1,095	5.6	8.25 GB	9	271
W_3	Bwa	Bioinformatics	1004	3.7	56.89 MB	3	1,000
W_4	Cycles	Agroecosystem	874	5.2	6.17 GB	4	432
W_5	1000Genome	Bioinformatics	328	6.0	25.96 GB	3	208
W_6	Blast	Bioinformatics	303	8.7	0.47 MB	3	300
W_7	Soykb	Bioinformatics	156	6.7	2.82 GB	11	100
W_8	Srsearch	Bioinformatics	22	5.2	16.50 GB	3	11

(i.e., execution time on a single 100Gflop/sec core) are in between 5 and 10 hours. The metrics shown in the table show that the workflow instances correspond to a diverse set of configurations, with different structures and different computation-data ratios. As a result, we expect that different scheduling algorithms will fare differently across these workflow instances.

The workflow instances available on the WfCommons collection do not include information about the execution of workflow tasks on multiple cores, but only gives a single execution time t , which is a sequential execution time on a single core. Thus, we assume an Amdahl’s Law parallel speedup behavior [2]: a task that executes in time t on one core executes in time $\alpha t/n + (1 - \alpha)t$ on n of these cores. For each task, we sample α uniformly between 0.8 and 1.0.

4.3 Algorithms

We assume that the WMS used to execute workflows employs a typical list-scheduling approach for deciding, at runtime, which ready task should be executed on which compute resources, while enforcing that not two tasks run simultaneously on the same core. The scheduling algorithm proceeds in three steps as follows. While there is at least one ready task and one idle core on which no task has been scheduled:

1. pick a ready task using some criterion C_1 ;
2. pick a cluster with at least one idle core using some criterion C_2 ;
3. pick a number of cores for the task execution using some criterion C_3 ;
4. schedule the picked task on the picked cluster using the picked number of cores.

We consider the following options for each of the above criteria:

– Criterion C_1 :

- 0: Pick the task with the largest bottom-level (i.e., prioritize tasks on the critical path);

- 1: Pick the task with the largest number of children tasks;
 - 2: Pick the task with the largest amount of input and output data;
 - 3: Pick the task with the largest amount of computation to perform.
- Criterion C_2 :
- 0: Pick the cluster that stores the largest amount of task input data in its network-attached storage;
 - 1: Pick the cluster with the most idle cores;
 - 2: Pick the cluster with the fastest cores.
- Criterion C_3 :
- 0: Pick as many cores as possible while ensuring that the task’s parallel efficiency is above 90%;
 - 1: Pick as many cores as possible while ensuring that the task’s parallel efficiency is above 50%;
 - 2: Pick as many cores as possible.

We denote each algorithm as A_x , where $x = 4 \times C_1 + 3 \times C_2 + 3 \times C_3$, which gives us 36 different algorithms (A_0 to A_{35}). All above scheduling criteria have been proposed in the literature. Although many other options could be considered, these 36 algorithms provide us with a sufficiently large and diverse sample set to conduct our investigation.

4.4 Experimental Methodology

An implementation of our online simulation approach in this case study entails (i) an implementation of a WMS that executes workflows on multi-cluster platforms; and (ii) an implementation of a simulator of these executions that can be invoked at runtime by the WMS. We face two main technical difficulties. First, to answer the third research question in Section 3 we need to experiment with different levels of simulation accuracy to measure the resulting impact on the effectiveness of our proposed approach, including quantifying the best-case effectiveness when online simulations are 100% accurate. This is not possible with a real-world implementation since a given simulator is necessarily inaccurate. Second, we wish to evaluate our approach on a large range of workflows, platforms, and algorithms. For instance, in this particular case study, we evaluate a total of $9 \times 8 \times 36 = 2,592$ experimental scenarios (9 platform configurations, 8 workflows, 36 algorithms). Even if we had access to a large number of different platform configurations, it would be difficult to obtain all experimental results, not only in terms of time and energy consumption, but also in terms of ensuring that these results are repeatable. The need to obtain many diverse and repeatable experimental results is, incidentally, the main reason why researchers in the field resort to simulation.

Given the above, we perform our case study entirely in simulation. We implement a WMS simulator, with WRENCH⁵ (v1.10) and SimGrid⁶ (v3.29), that simulates a WMS that executes workflows on multi-cluster platforms using any

⁵ <https://wrench-project.org>

⁶ <https://simgrid.org>

one of our 36 algorithms. This simulator provides us with an analog of a production WMS implementation, which we enhance with our online simulation approach. That is, during its simulated execution, our simulator runs as many (online) simulations of its future execution as there are scheduling algorithms (36 in this case study). This is done simply by having the simulator call the `fork` system call to create a child process that is a clone of the simulator, for each algorithm. Each child then continues the simulated workflow execution and reports the simulated workflow completion date to its parent process. In this fashion, the simulator can explore all its possible futures for all algorithms. The WMS then picks the algorithm that achieved the fastest workflow execution in those simulations. The simulator outputs the workflow makespan, in seconds, based on the following input:

- **A workflow instance** – One of the 8 instances in Section 4.2, available as a JSON file using the WfFormat format⁷. We use w to denote the total amount of sequential work, i.e., the sum of the sequential task execution times on a reference 100Gflops/sec core (the 4th column in Table 2).
- **A platform configuration** – One of the 9 configurations in Section 4.1.
- **A fraction of total work, α** – This parameter defines how often our online simulation approach is applied throughout workflow execution: it is applied at the onset of the workflow execution and subsequently each time an additional fraction α of the total work w has been completed. For instance, $w = 10,000$ Gflop and $\alpha = 0.2$, our approach will be invoked 5 times throughout workflow execution, once at the beginning of the execution, and once each time an additional 2,000 Gflop of sequential work has been performed.
- **A fraction of total work, β** – Each online simulation proceeds until execution of a fraction β of the total sequential work has been simulated and reports the current simulation date to the parent process. In other terms, β defines the time horizon of the simulations.
- **A relative simulation error, e** – This parameter denotes the relative range of an uniformly distributed simulation error. That is, when an online simulation determines that a fraction β of the sequential work was performed in time t , it reports, instead, a time $\max(0, t + \mathcal{U}(-t \times e, t \times e))$, where $\mathcal{U}(a, b)$ denotes the uniform random distribution on the (a, b) interval. For any experiment for which $e > 0$, we run 10 samples.

Simulator code and all simulation data are publicly available⁸.

5 Results

5.1 Diversity of one-algorithm approaches

In Section 4, we claimed that our experimental scenarios (workflow and platform configurations) would lead our different algorithms to exhibit a range of behav-

⁷ <https://wfcommons.org/format>

⁸ https://github.com/wrench-project/jssp2022_submission_data

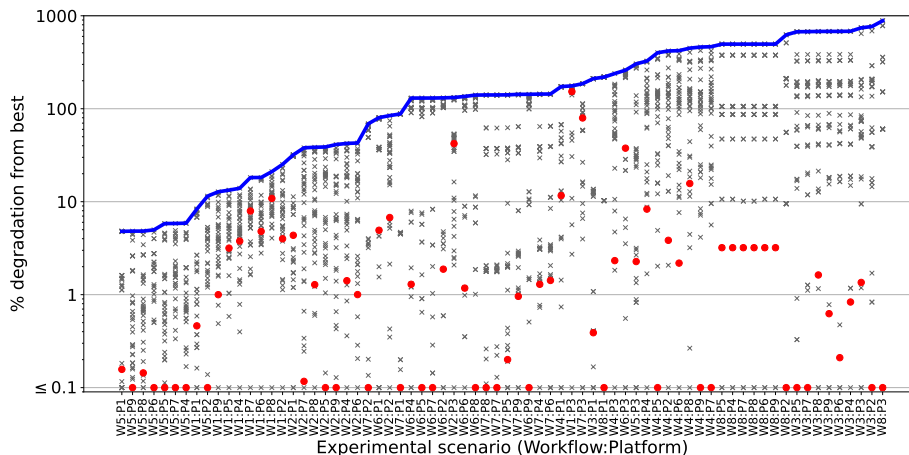


Fig. 1: Percent degradation from best for all algorithms over all experimental scenarios, sorted by increasing maximum values. Maximum values are shown as a blue solid lines. Data points for the A_8 algorithm are shown as red dots.

iors. In this section, we verify this claim quantitatively. Figure 1 shows, for each experimental scenario (i.e., a workflow and platform combination) the relative difference, in percentage, between the makespan achieved by each algorithm and that achieved by the best algorithm for this scenario, which is typically termed “degradation from best” or *dfb*. The scenarios are sorted by increasing value of the maximum *dfb*. Results show that maximum *dfb* values range from 4.38% to 883.81%. We note that the experimental scenarios on the horizontal axis are loosely sorted by the workflow configurations, meaning that scheduling algorithm behaviors are sensitive to workflow structures. Furthermore, we see that for most experimental scenarios, many algorithms lead to different *dfb* values, and thus makespans. Overall, we conclude that our experimental scenarios are sufficient to highlight the diversity between our 36 scheduling algorithms.

Although the above results indicate diversity, one may wonder whether some (or perhaps just one?) algorithm is always best, in which case, one should just use that algorithm. To this end, for each algorithm, we can compute its average *dfb* over all experimental scenarios. We find that algorithm A_8 achieves the lowest average *dfb* at 6.47%. While this number is relatively low, it does not mean that algorithm A_8 is consistently a good choice. It happens to be the best (or within 1% of the best) choice for 37 of our 72 scenarios. However, it has a *dfb* higher than 10% for 7 of the remaining 35 scenarios, and as high as 159.60%. This is illustrated in Figure 1 where the data points for algorithm A_8 are shown as red dots. We conclude that no single algorithm is best, and that although algorithm A_8 is the “best on average” choice, it can be vastly outperformed by other algorithms for some experimental scenarios.

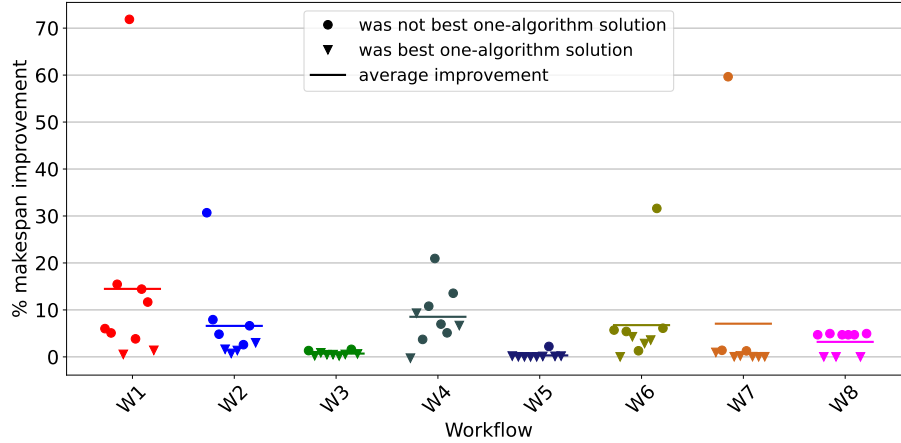


Fig. 2: Percentage improvement over the one-algorithm approach for each workflow (each data point is for a different platform configuration).

5.2 Evaluation in the ideal case ($\beta = 1$, $e = 0$)

In this section, we compare our online simulation approach to the one-algorithm approach under ideal conditions, that is, with the two following assumptions: (i) each online simulation simulates the application execution until completion ($\beta = 1$); and (ii) simulations are 100% accurate ($e = 0$). In upcoming sections, we relax these assumptions. Unless specified otherwise, all results hereafter are obtained with $\alpha = 0.1$, i.e., online simulations are invoked 10 times throughout workflow execution.

Because of these two assumptions, given any experimental scenario, our approach is guaranteed to never be outperformed by any one algorithm: at the onset of the execution it simulates all possible algorithms and necessarily picks the best one. That is, if we were to plot the degradation from best of our approach in Figure 1, its data points would all be on the $y = 0$ line. In this and upcoming sections, we compare our approach to the one-algorithm approach that uses algorithm A_8 , which, for simplicity, we term the *one-algorithm approach*. As seen in the previous section, A_8 is the algorithm with the lowest average degradation from best among all 36 algorithms. It thus corresponds to the best choice that a runtime system developer could make if asked to pick one algorithm to implement in their system, at least in the scope of this case study. Picking A_8 as our main competitor allows us to evaluate the effectiveness of our approach in the worst case. We note that, in practice, the runtime system developer may very well pick another algorithm, in which case all results hereafter would be more favorable (and often drastically more favorable) for our approach.

Figure 2 shows relative makespan improvements over the one-algorithm approach. Results are grouped by workflow, showing 9 data points for each workflow (for the 9 platform configurations). Horizontal lines show average improvements.

Relative improvement is always positive and can be large, and average improvement is above 5% for 5 of the 8 workflow configurations (Table 2).

Two kinds of data points are shown in Figure 2. The data points marked with circles correspond to cases in which A_8 is not the best, or close to the best, of the 36 algorithms for that experimental scenario (i.e., its degradation from best is larger than 1%). For these data points, we expect our proposed approach to provide improvement because it will simply use another algorithm. For instance, the data point above 70% for workflow W_1 corresponds to an execution on platform P_3 . For this experimental scenario, Figure 1 shows that algorithm A_8 has almost the worst degradation from best. Our approach thus eliminates A_8 from consideration based on simulation results.

The data points marked with triangles correspond to experimental scenarios in which A_8 has degradation from best below 1%. For some of these scenarios our approach leads to non-negligible improvement (up to 9.3% improvement for the $W_4:P_5$ scenario). This is because, for these scenarios, it is beneficial to use more than one scheduling algorithm. In fact, we can compare our approach to a one-algorithm “oracle” that would always pick the best algorithm to use for each experimental scenario. We find that our approach outperforms this oracle for 56 of our 72 experimental scenarios, and outperforms it by more than 5% for 11 of them. The main motivation for this work is that it is difficult to pick one algorithm to implement as part of a CI runtime system. These results show that one should, in fact, use more than one algorithm for a single workflow execution.

An interesting question is that of the number of different algorithms used by our approach. In these results, this number is at most 10 since $\alpha = 0.1$. Our approach uses a single algorithm for only 4 of our 72 experimental scenarios. Across all scenarios, our approach uses up to 6 different algorithms during a single workflow execution and 3.08 different algorithms on average. Overall, out of our 36 different algorithms 25 of them end up being used at least once by our approach. Algorithm A_8 is, unsurprisingly, the algorithm most used by our approach. But some algorithms that have poor average degradation from best are also used. For instance, algorithm A_0 is used for 12 of our 72 scenarios, but has the 4th largest average degradation from best at 176.26%.

5.3 Evaluation with shorter simulation time horizons ($\beta < 1$)

One may wonder whether it is necessary for online simulations to simulate the execution until completion. The results in the previous section are for $\beta = 1$, i.e., workflow execution is always simulated until completion. Given a value of β , a workflow, and a platform configuration, we measure the percentage improvement (or loss) that our approach achieves over the one-algorithm approach. Figure 3 shows results for several β value and workflow combinations. For each combination, there are 9 data points, one for each platform configuration. For better readability, these data points are shown as violin plots, which indicate the minimum, maximum, and average values as well as the shape of the distribution. Each data point below the $y = 0$ line corresponds to cases in which our approach loses to the one-algorithm approach.

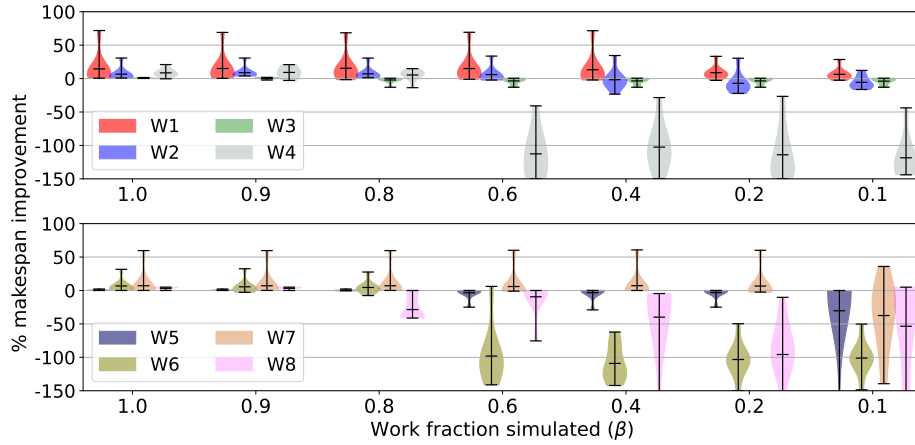


Fig. 3: Percentage improvement over the one-algorithm approach, for each workflow and for different β values. Each violin plot shows minimum, maximum, and average values as well as the overall shape of the distribution of the data points.

As expected, the results in Figure 3 show that the number of times our approach loses to the one-algorithm approach increases as β decreases, i.e., as the simulation becomes more shortsighted. But the trends vary depending on the workflow. At one extreme, e.g. for workflow W_1 , our approach remains beneficial for β as low as 0.1 (i.e., when only 10% of the total work is simulated). At the other extreme, for workflow W_8 , as soon as β is 0.8 or below, our approach experiences losses. The fact that different workflows exhibit different behaviors is not surprising. Depending on workflow structures, scheduling decisions made at the onset of the execution may or may not have a large influence on the later phases of that execution. Given that, it is likely difficult to determine what level of shortsightedness is acceptable for a given workflow. We then conclude that simulating the entire execution of the application until completion ($\beta = 1$) is the safest option. All results presented hereafter, unless specified otherwise, are for $\beta = 1$. The downside of using $\beta = 1$ is that it maximizes simulation times, the implications of which are discussed in Section 5.5.

5.4 Evaluation with simulation inaccuracies ($e > 0$)

Simulations are never accurate, and we need to ascertain whether our approach can tolerate simulation inaccuracies. To answer this question, we apply uniformly distributed perturbations to simulated makespans in the interval $[-e, e]$, for various values of e (see details in Section 4.4). Figure 4 is similar to Figure 3 but shows results for several values of e . For $e > 0$, each violin plot in the figure corresponds to 90 data points (9 platform configurations and 10 samples for 10 different seeds of the random number generator). Results show that our approach is reasonably tolerant to simulation error. Even when $e = 0.2$ (i.e., a simulated

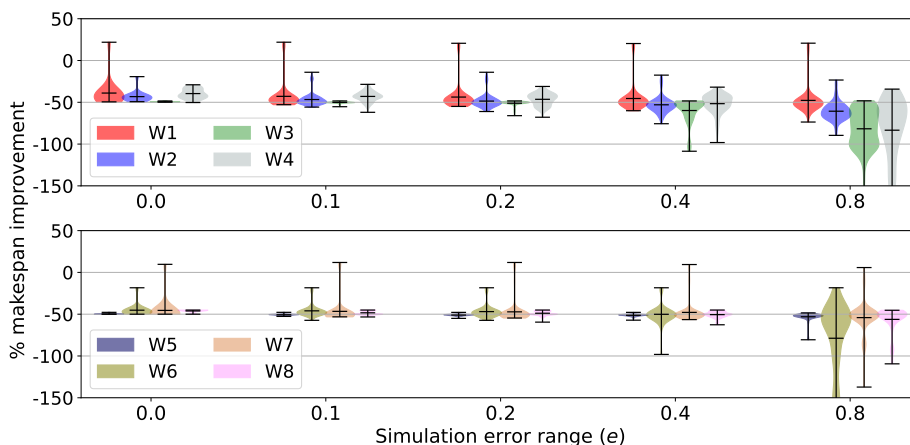


Fig. 4: Percentage improvement over the one-algorithm approach, for each workflow and for different e values. Each violin plot shows minimum, maximum, and average values as well as the overall shape of the distribution of the data points.

makespan can be underestimated or overestimated by up to 20%), our approach remains mostly beneficial and maintains positive average improvement over the one-algorithm approach for all workflows. For $e = 0.4$ and above, our approach begins to be outperformed by the one-algorithm approach.

Simulators developed using SimGrid and WRENCH, as the one developed in this work, have been reported to achieve simulation errors well below 20%. For instance, the WMS simulator in [10] achieves makespan errors below 5%. Other simulators, however, may experience higher error. In practice, it would then be useful to perform *simulation error forensics* and apply corrective measures. That is, the runtime system could keep track of the simulated execution for the algorithm that ends up being selected, and then compare this execution to what actually happened in the real execution. The goal would be to identify sources of simulation error, and correct for them in the instantiation of the simulator before the next round of online simulations.

Overall, we conclude that simulation errors with current state-of-the-art simulation implementations, albeit unavoidable, are sufficiently small or mitigable for our online simulation approach to be feasible.

5.5 Simulation overhead

Some parallel and distributed computing simulation frameworks, such as SimGrid, which we use in this work, have placed a large emphasis on scalability. To this end, analytical simulation models have been developed that have low computational complexity and that can be implemented efficiently. In Section 5.3, we saw that it is typically useful to simulate the upcoming application execution to completion. Furthermore, the number of algorithms to simulate could

(and should) be large. Therefore, one concern for the feasibility of our approach in practice is the overhead of running simulations, in spite of these simulations relying on scalable simulation models.

Table 3: Simulated makespan, simulation time, ratio thereof, and peak memory footprint of the simulation when simulating the execution of each workflow on platform configuration P_4 with algorithm A_8 . Results obtained on a 2.3GHz core.

Workflow	simulated makespan (sec)	simulation time (sec)	ratio	peak memory footprint (MB)
W_1	338.77	29.35	11.5	149.95
W_2	221.67	2.86	77.5	36.19
W_3	170.63	5.58	30.6	65.98
W_4	57.62	4.48	12.9	65.58
W_5	5,618.07	3.20	1,755.6	16.96
W_6	57.21	0.77	74.3	19.32
W_7	4,887.52	6.97	701.2	28.96
W_8	416.16	0.11	3783.2	5.98

Most simulation frameworks implement discrete-event (as opposed to discrete-time) simulation. That is, computational complexity depends on the number of events to simulate and not on the length of time being simulated. Table 3 shows results obtained when simulating the full execution of each workflow on platform configuration P_4 using algorithm A_8 . Simulations were executed on one core of a 2.3GHz Intel Core i9 and the results in the table are averaged over 10 trials. Since algorithm A_8 generally leads to shorter makespans than its competitors, the results in the table correspond to a worst case in terms of ratio of simulated makespan to simulation time. Also note that these results are for simulating the full workflow execution. As the execution progresses, online simulations only need to simulate the remaining application execution. That is, the simulation overhead decreases at each round of online simulation. Thus the results in the table correspond to the maximum (initial) simulation overhead.

We find that for most workflows the ratio of simulated makespan to simulation time is large. But for some workflows, such as W_1 , the ratio is only 11.5x. This is because this workflow has a high number of tasks relative to its total computational work as well as a high data footprint (see Table 2), which increases the number of execution events to simulate. This is also the case for workflow W_4 , and in this case is also due to the fact that the simulated makespan is low. As seen in Figure 3, for these two workflows, it would be possible to reduce the fraction of work being simulated, so as to reduce the simulation time. In particular, our proposed approach performs well for W_1 even when simulating the execution of only 10% of the total work. Regardless, a key point here is that running online simulations does not have to hold up the application execution, but can be done concurrently with that execution. That is, the online simulation overhead can be fully hidden by the application execution.

The results in Table 3 are for the simulation of one algorithm. Our approach needs to run one simulation for each available algorithm (36, in this case study). These simulations are independent and can be executed in parallel on multiple cores, which is feasible due to the relatively low memory footprints reported in Table 3. For instance, running 36 concurrent simulations for workflow W_1 , which causes the largest simulation memory footprint in our case study, only requires 5.2GB of RAM. Running these 36 simulations concurrently on a 48-core Cascadelake 2.8GHz machine takes only 23% longer than running only the slowest one of these simulations (simulations take different amounts of time depending on the scheduling algorithm in use).

Another option for mitigating simulation overhead is to reduce the frequency at which online simulations are executed. All experiments presented so far have used $\alpha = 0.1$, that is, online simulations are invoked each time 10% of the total work has been completed. It turns out that, at least for the results in this case study, increasing α does not lead to significant performance degradation. We conducted experiments with $\alpha = 0.2$, so that online simulations are invoked only 5 times during the whole execution instead of 10 times with $\alpha = 0.1$. Comparing results between our approach and the one-algorithm approach, we find that there is at most a one-point decrease in effectiveness for 7 of the workflows and at most a two-point decrease for the remaining workflow. That is, if with $\alpha = 0.1$ our approach outperforms the one-algorithm approach by $x\%$, then with $\alpha = 0.2$ it outperforms it by at least $x - 2\%$ and typically by at least $x - 1\%$. In no instance does our approach lose to the one-algorithm approach with $\alpha = 0.2$. These results are obtained assuming that simulations are perfectly accurate. For a simulation error range at 20% ($e = 0.2$), then our our approach experiences less than a one-point decrease in effectiveness for 5 workflows (instead of 7) and less than a two-point decrease for the remaining 3 workflows (instead of 1). Overall, at least within the scope of this case study, decreasing the frequency at which online simulations are executed, which reduces simulation overhead, does not have a large negative impact on the overall effectiveness of our approach.

6 Conclusion

In this work, we have assessed the potential merit of using online simulations to drive the scheduling decisions made by CI runtime systems. The main goal of the approach is to obviate the well-known challenge of selecting a particular scheduling algorithm to implement in a runtime system. This challenge stems not only from the sheer number of proposed algorithms, but also from the fact that the effectiveness of these algorithms is often unclear in practice and/or on scheduling problem instances at hand. Our case study has shown that our proposed approach outperforms the one-algorithm approach, even if this approach happens to use the algorithm that performs best, on average, across all experimental scenarios in the case study. Although in some cases online simulations remain effective when simulating only a fraction of the upcoming execution, simulating the execution to completion seems like the safest option. Crucially, our

proposed approach retains its advantage over the one-algorithm approach even in the presence of relatively large simulation error, i.e., larger than what state-of-the-art simulators have been reported to achieve. Finally, we have discussed how simulation overhead can be mitigated using a variety of techniques.

Recall that we have compared our proposed approach to the best possible rational choice a runtime system developer could make for implementing the one-algorithm approach in the context of our case study (i.e., pick algorithm A_8). It is not clear how this best choice could be made in practice (besides by conducting a full experimental case study as done in this work), hence the main motivation for this work. Were the system developer to pick a middle-of-the-pack algorithm, say algorithm A_{22} , which has an average degradation from best at 49.79% (the worst algorithm has average degradation from best at 179.23%), all results presented in Section 5 would be drastically improved. For instance, our approach would outperform the one-algorithm approach on average for all workflows for simulation error ranges up to 80% (instead of up to 20%).

The results in this work show that it is likely worth implementing our approach in a real runtime system. We plan to do so as part of production Workflow Management Systems, so that we can reproduce in practice some of the results presented in our case study. A particularly interesting future work direction, to be pursued once a prototype implementation is available, is the investigation of simulation forensics techniques to detect and mitigate simulation error at runtime. Finally, if willing to use a large number of candidate scheduling algorithms, i.e., well beyond the 36 used in our case study, it may also be necessary to investigate techniques for pruning the set of candidate algorithms (removing algorithms that tend to perform similarly, removing algorithms that tend to perform poorly) to avoid prohibitive simulation overhead.

Acknowledgments

This work is funded by NSF contracts #2106059 and #2106147: “Collaborative Research: OAC Core: Simulation-driven runtime resource management for distributed workflow applications”; and partially funded by NSF contracts #2103489 and #2103508. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. Finally, we thank the NSF Chameleon Cloud for providing time grants to access their resources.

References

1. Adhikari, M., Amgoth, T., Srirama, S.N.: A survey on scheduling strategies for workflows in cloud environment and emerging trends. *ACM Computing Surveys (CSUR)* **52**(4), 1–36 (2019)
2. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: *Proceedings of the April 18-20, 1967, spring joint computer conference*. pp. 483–485 (1967)

3. Arya, L.K., Verma, A.: Workflow scheduling algorithms in cloud environment - A survey. In: Proc. of Conf. on Recent Advances in Engineering and Computational Sciences (2014)
4. Badia Sala, R.M., Ayguadé Parra, E., Labarta Mancho, J.J.: Workflows for science: A challenge when facing the convergence of HPC and big data. *Supercomputing frontiers and innovations* **4**(1), 27–47 (2017)
5. Buyya, R., Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience* **14**(13-15), 1175–1220 (Dec 2002)
6. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and Experience* **41**(1), 23–50 (Jan 2011)
7. Carastan-Santos, D., de Camargo, R.Y.: Obtaining Dynamic Scheduling Policies with Simulation and Machine Learning. In: Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '17, Association for Computing Machinery, New York, NY, USA (2017)
8. Carothers, C.D., Bauer, D., Pearce, S.: ROSS: A High-Performance, Low Memory, Modular Time Warp System. In: Proc. of the 14th ACM/IEEE/SCS Workshop of Parallel on Distributed Simulation. pp. 53–60 (2000)
9. Casanova, H., Giersch, A., Legrand, A., Qinson, M., Suter, F.: Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. *Journal of Parallel and Distributed Computing* **75**(10), 2899–2917 (2014)
10. Casanova, H., Ferreira da Silva, R., Tanaka, R., Pandey, S., Jethwani, G., Koch, W., Albrecht, S., Oeth, J., Suter, F.: Developing accurate and scalable simulators of production workflow management systems with wrench. *Future Generation Computer Systems* **112**, 162–175 (2020)
11. Coleman, T., Casanova, H., Pottier, L., Kaushik, M., Deelman, E., Ferreira da Silva, R.: Wfcommons: A framework for enabling scientific workflow research and development. *Future Generation Computer Systems* **128**, 16–27 (2022)
12. Eyraud-Dubois, L., Legrand, A.: The Influence of Platform Models on Scheduling Techniques. In: Robert, Y., Vivien, F. (eds.) *Introduction to Scheduling*, chap. 11, pp. 281–309. CRC Press (2009)
13. Gupta, A., Garg, R.: Workflow scheduling in heterogeneous computing systems: A survey. In: 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN). pp. 319–326. IEEE (2017)
14. Hoefler, T., Schneider, T., Lumsdaine, A.: LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In: Proc. of the ACM Workshop on Large-Scale System and Application Performance. pp. 597–604 (Jun 2010)
15. Kecskemeti, G.: DISSECT-CF: A simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory* **58**(2), 188–218 (2015)
16. Kecskemeti, G., Ostermann, S., Prodan, R.: Fostering Energy-Awareness in Simulations Behind Scientific Workflow Management Systems. In: Proc. of the 7th IEEE/ACM Intl. Conf. on Utility and Cloud Computing. pp. 29–38 (2014)
17. Liu, J., Pacitti, E., Valduriez, P., Mattoso, M.: A Survey of Data-Intensive Scientific Workflow Management. *J. Grid Comput.* **13**(4), 457–493 (2015)
18. Malik, A.W., Bilal, K., Aziz, K., Kliazovich, D., Ghani, N., Khan, S.U., Buyya, R.: Cloudnetsim++: A toolkit for data center simulations in omnet++. In: Proc.

- of the 2014 11th Annual High Capacity Optical Networks and Emerging/Enabling Technologies (Photonics for Energy). pp. 104–108 (2014)
19. Nallakumar, R., Sruthi Priya, K.: A Survey on Deadline Constrained Workflow Scheduling Algorithms in Cloud Environment. *International Journal of Computer Science Trends and Technology* **2**(5), 44–50 (2014)
 20. Nazarenki, A., Sukhoroslov, O.: Using Simulation to Improve Workflow Scheduling in Heterogeneous Computing Systems. In: *Proc. of Russian Supercomputing Days*. pp. 480–490 (2017)
 21. Núñez, A., Vázquez-Poletti, J., Caminero, A., Carretero, J., Llorente, I.M.: Design of a New Cloud Computing Simulation Platform. In: *Proc. of the 11th Intl. Conf. on Computational Science and its Applications*. pp. 582–593 (June 2011)
 22. Pérez, E.: A simulation-driven online scheduling algorithm for the maintenance and operation of wind farm systems. *SIMULATION* **98**(1), 47–61 (jul 2021)
 23. Qayyum, T., Malik, A.W., Khan Khattak, M.A., Khalid, O., Khan, S.U.: FogNet-Sim++: A Toolkit for Modeling and Simulation of Distributed Fog Environment. *IEEE Access* **6**, 63570–63583 (2018)
 24. Rodriguez, M.A., Buyya, R.: A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments. *Concurrency and Computation: Practice and Experience* **29**(8), e4041 (2017)
 25. Sfiligoi, I.: Estimating job runtime for CMS analysis jobs. *Journal of Physics: Conference Series* **513**(3), 032087 (2014)
 26. Ferreira da Silva, R., Casanova, H., Chard, K., Altintas, I., Badia, R.M., Balis, B., Coleman, T.a., Coppens, F., Di Natale, F., Enders, B., Fahringer, T., Filgueira, R., Fursin, G., Garijo, D., Goble, C., Howell, D., Jha, S., Katz, D.S., Laney, D., Leser, U., Malawski, M., Mehta, K., Pottier, L., Ozik, J., Peterson, J.L., Ramakrishnan, L., Soiland-Reyes, S., Thain, D., Wolf, M.: A community roadmap for scientific workflows research and development. In: *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*. pp. 81–90 (2021)
 27. Ferreira da Silva, R., Rynge, M., Juve, G., Sfiligoi, I., Deelman, E., Letts, J., Würthwein, F., Livny, M.: Characterizing a high throughput computing workload: The compact muon solenoid (CMS) experiment at LHC. *Procedia Comp. Sci.* **51**, 39–48 (2015), international Conference On Computational Science (ICCCS)
 28. Singh, L., Singh, S.: A Survey of Workflow Scheduling Algorithms and Research Issues. *International Journal of Computer Applications* **74**(15), 21–28 (2013)
 29. Sinnen, O.: *Task Scheduling for Parallel Systems (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, USA (2007)
 30. Tikir, M., Laurenzano, M., Carrington, L., Snavely, A.: PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications. In: *Proc. of the 15th Intl. Euro-Par Conf. on Parallel Processing*. pp. 135–148. No. 5704 in LNCS, Springer (Aug 2009)
 31. Velho, P., Mello Schnorr, L., Casanova, H., Legrand, A.: On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations. *ACM Transactions on Modeling and Computer Simulation* **23**(4) (2013)
 32. Existing workflow systems. <https://s.apache.org/existing-workflow-systems> (2022)