Multiple Workflows Scheduling in Multi-tenant Distributed Systems: A Taxonomy and Future Directions

MUHAMMAD H. HILMAN, MARIA A. RODRIGUEZ, and RAJKUMAR BUYYA, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia

Workflows are an application model that enables the automated execution of multiple interdependent and interconnected tasks. They are widely used by the scientific community to manage the distributed execution and dataflow of complex simulations and experiments. As the popularity of scientific workflows continue to rise, and their computational requirements continue to increase, the emergence and adoption of multi-tenant computing platforms that offer the execution of these workflows as a service becomes widespread. This article discusses the scheduling and resource provisioning problems particular to this type of platform. It presents a detailed taxonomy and a comprehensive survey of the current literature and identifies future directions to foster research in the field of multiple workflow scheduling in multi-tenant distributed computing systems.

CCS Concepts: • **Information systems → Computing platforms**; • **Networks → Cloud computing**; • **Applied computing → Service-oriented architectures**; • **Computer systems organization → Cloud computing**; **Grid computing**;

Additional Key Words and Phrases: Scientific workflows, multi-tenant platforms, multiple workflows scheduling

ACM Reference format:
Muhammad H. Hilman, Maria A. Rodriguez, and Rajkumar Buyya. 2020. Multiple Workflows Scheduling in Multi-tenant Distributed Systems: A Taxonomy and Future Directions. *ACM Comput. Surv.* 53, 1, Article 10 (February 2020), 39 pages.
https://doi.org/10.1145/3368036

1 INTRODUCTION

The concept of workflow has evolved from the manufacturing and business process fields to a broader notion representing a structured process flow design. Within the scientific community, workflows refer to a model for automating and managing the distributed execution of a complex scientific problem that requires the flow of data among different applications [1]. Many scientific problems adopt this model to overcome the limitations of an individual scientific application to process a vast amount of data. Examples of this type of workflow include Scientific Workflows

This research is partially supported by the LPDP (Indonesia Endowment Fund for Education) and Australia Research Council research grant.
Authors' address: M. H. Hilman, M. A. Rodriguez, and R. Buyya, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Parkville, Melbourne, VIC, 3000, Australia; emails: hilmanm@student.unimelb.edu.au, {maria.rodriguez, rbuyya}@unimelb.edu.au.

0360-0300/2020/02-ART10 $15.00
https://doi.org/10.1145/3368036

Table 1. Resume on Existing Taxonomy of Workflow Scheduling Algorithms

Existing Works	Review Types		Environments		Workloads		Focus of Discussion
	Survey	Systematic	Grids	Clouds	Single	Multiple	
Yu and Buyya [4]	✓	-	✓	-	✓	-	Workflow management system architecture
Wieczorek et al. [5]	✓	-	✓	-	✓	-	Multiple criteria on workflow scheduling
Wu et al. [6]	✓	-	-	✓	✓	limited	Workflow scheduling in cloud environments
Singh and Chana [7]	-	✓	-	✓	✓	limited	Resource scheduling in cloud environments
Alkhanak et al. [8]	-	✓	-	✓	✓	limited	Cost-aware scheduling in cloud environments
Smanchat and Viriyapant [9]	✓	-	-	✓	✓	-	Workflow scheduling in cloud environments
Rodriguez and Buyya [10]	✓	-	-	✓	✓	limited	Workflow scheduling in cloud environments
Our current work	✓	-	✓	✓	-	✓	Multi-tenancy in multiple workflows scheduling

[2] that consist of HPC applications for e-Science and MapReduce Workflows [3] that are used to process Big Data analytics. Such workflows are large-scale distributed applications and require extensive computational resources to execute in a reasonable amount of processing time. Therefore, workflows are commonly deployed on distributed systems, in particular, cluster, grid, and cloud computing environments.

Workflow scheduling was studied and surveyed extensively during the cluster and grid computing era [4, 5]. Subsequently, when cloud computing emerged as a new paradigm with market-oriented focus, the scientific community got a promising deployment platform for workflow applications offering multiple benefits. However, this emerging paradigm also brought forth additional challenges. Solutions for cloud workflow scheduling have been extensively researched, and a variety of algorithms have been developed [6, 7]. Furthermore, various existing taxonomies of workflow scheduling in clouds focus on describing the particular scheduling problem as well as its unique challenges and ways of addressing them [8–10]. The summary of these relevant works is presented in Table 1 along with the highlight focus of the papers.

Contrary to these previous studies that focus mostly on a single workflow scheduling, this study addresses the scheduling problem from a higher-level perspective. It considers the scheduling of multiple workflows that arrive continuously to a multi-tenant computing platform. The advent of multi-tenant environments like clouds and the shifting trend from the traditional on-premises to the utility computing era has led to the emergence of platforms that offer workflows processing as a service. These platforms continuously receive many requests for workflow executions from different users, along with their various Quality of Service (QoS) requirements. The provider must then be able to schedule these workflows in a way that each of their requirements is fulfilled. A simple way to achieve this is by allocating a set of dedicated resources to execute each workflow. However, the inter-dependent tasks produce unavoidable idle gaps in the schedule. Hence, dedicating a set of resources for each user can be considered inefficient in environments where multiple workflows are involved, as it leads to resources being underutilized. This approach, in turn, may cause a significant loss for the providers that generate revenue from the utilization of resources. Consequently, the strategies implemented in such platforms should aim to improve resource utilization while still complying with the unique requirements of different users.

Accommodating multi-tenants with different requirements creates a high-complexity management system. The very first problem lies on how such a system handles the various workflow applications. A variety of applications involve different software libraries, dependencies, and hardware requirements. The users should be able to customize the specific configurations along with their defined QoS when submitting the workflows. Furthermore, multi-tenant systems must have a general scheduling approach to handle different types of computational requirements from different workflows. Another consideration related to multi-tenancy is the strategy to maintain fairness between multiple users that should be achieved through clear prioritization in the scheduling and the automatic scaling of the resources. The last aspect that should be noticed in multi-tenant computing platforms is the performance variability in computational resources as virtualization-based infrastructures like clouds may encounter performance degradation due to the multi-tenancy, virtualization overhead, geographical location, and temporal aspects [11].

The contribution of this work is the study of the multiple workflows scheduling problems in multi-tenant distributed computing systems. The structure of the rest of this article is as follows. Section 2 discusses the multiple workflows scheduling problems, while Section 3 addresses its relevancy in multi-tenant computing platforms. The proposed taxonomy is presented in Section 4, and the review of existing solutions is covered in Section 5, along with their classification. Section 6 offers future directions, and Section 7 summarizes the article.

2 SCHEDULING MULTIPLE WORKFLOWS IN MULTI-TENANT ENVIRONMENTS

The scope of this work is limited to the theoretical scheduling algorithms for multiple workflows that are modeled into directed acyclic graphs (DAGs) where a workflow W consists of a set of tasks $T = (t_1, t_2, \ldots, t_n)$ and a set of directed edges $E = (e_{12}, e_{13}, \ldots, e_{mn})$ in which an edge e_{ij} represents a data dependency between task t_i (parent task) and task t_j (child task). Hence, t_j will only be ready for execution after t_i has completed. The purpose of DAG scheduling is allocating the tasks to computational resources in a way that the precedence constraints among the tasks are preserved. Specifically, the workflows submitted to multi-tenant computing platforms belong to different users and are not necessarily related to each other. As a result, heterogeneity becomes a defining characteristic of the workload that covers various aspects of workflows, including the type of applications, the size of the workflows, and the user-defined QoS.

When these workflows arrive into the platforms, planning the schedule by exploiting the information (i.e., topological structure, computational requirement, size, input) as being implemented in static workflow scheduling is not always beneficial. For example, a strategy of partitioning tasks before runtime to minimize the data transfer is proven to be efficient for data-intensive workflows [12]. This strategy can be adopted in multi-tenant computing platforms, but then, it faces an inevitable bottleneck, since the time required for partitioning a workflow may delay the next queue of arriving workflows for scheduling. The waiting time may significantly increase when involving a metaheuristic technique—which is known for its compute-intensive pre-processing—in the planning phase. As the size of the workflow increases, this pre-processing time may become longer and produce a more massive queue with significant waiting time delay. Hence, we do not consider solutions that schedule each workflow independently, as depicted in Figure 1(a), as this approach is no different from scheduling a single workflow.

Instead, we consider scheduling algorithms designed to schedule multiple workflows simultaneously, as shown in Figure 1(b). Within this context, the multiple workflows scheduling problem that is being addressed focus on how to allocate the tasks $T = (t_1, t_2, \ldots, t_n)$ from different workflows $W = \{w_1, w_2, w_3, \ldots, w_n\}$ on a multi-tenant distributed computing systems with several computational resources $R = \{r_1, r_2, r_3, \ldots, r_n\}$ so that the idle time slots generated from executing a workflow w_i on a resource r_i can be allocated to the task t_j from another workflow w_j. There are many benefits and challenges of scheduling multiple workflows in this approach. The main benefits are related to re-using and sharing of the idle time slots and reduction of waiting time from queueing delay of workflows being scheduled. However, the challenges to achieving these are not trivial. Handling the workloads heterogeneity, managing the continuous arriving workflows, implementing general scheduling approaches that deal with different requirements, and dealing with performance variability in distributed systems are questions that must be answered.

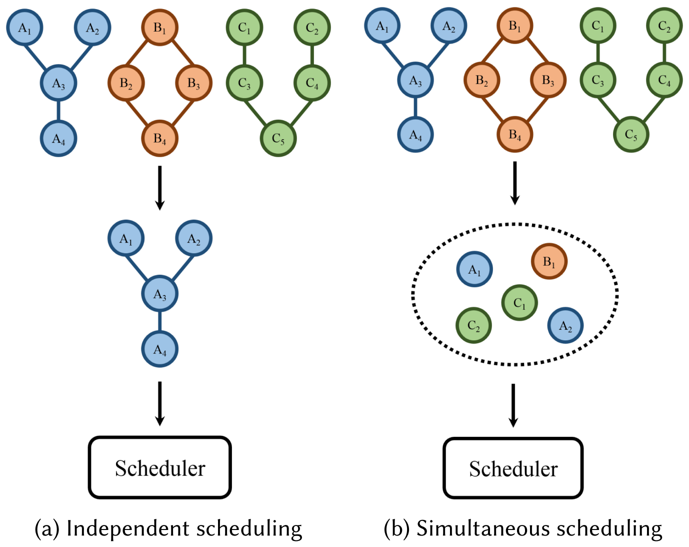

Fig. 1. Scheduling multiple workflows.

3 WORKFLOW AS A SERVICE PLATFORM FOR SCIENTIFIC APPLICATIONS

Scientific workflows are widely used to automate scientific experiments in many areas. The latest detection of gravitational waves by the LIGO project [13] is an example of a scientific breakthrough assisted by workflow technologies. These workflows are composed of multiple tasks and dependencies that represent the flow of data between them. Scientific workflows are usually large-scale applications that require extensive computational resources to process. As a result, distributed systems with an abundance of storage, network, and computing capacity are widely used to deploy these resource-intensive applications.

The complexity of scientific workflows execution urges scientists to rely on workflow management systems (WMS), which manage the deployment of workflows in distributed resources. Their main functionalities include but are not limited to scheduling the workflows, provisioning the resources, managing the dependencies of the tasks, and staging the input/output data. A key responsibility of WMS and the focus of this work is the scheduling of workflow tasks. In general, this process consists of two stages: (i) mapping the execution of tasks on distributed resources and (ii) acquiring and allocating the appropriate compute resources to support them. Both of these processes need to be carried out while considering the QoS requirements specified by users and preserving the dependencies between tasks. These requirements make the workflow scheduling process a challenging problem.

WMS technology has been consistently evolved along with the emergence of multi-tenant distributed computing systems. ASKALON [14], Cloudbus [15], HyperFlow [16], Kepler [17], Pegasus [18], and Taverna [19] are examples of the WMS that continuously developed within the scientific community. In the era of cluster and grid environments, the main features of the WMS are focusing on the workflow's representation, utilizing web technology, and providing the Graphical User Interface for the users. Then, the requirements shifted to the fulfillment of user-defined QoS in market-oriented clouds, followed by the need to make workflow's execution more lightweight using various microservices technology.

The advancement of e-Science infrastructure in the form of scientific applications empowers a large number of scientists around the world to start the shifting trend of scientific experiments.

They are part of the broad community that is called the *Long Tail of Science*. The smaller group of scientists that run the scientific experiments on a far tinier scale than the LIGO project but generate a more significant number of scientific data and findings [20]. However, not many scientists can afford to build their dedicated computational infrastructure for their experiments. In this case, there is a potential market for providing such services to the scientific community.

Workflow as a Service (WaaS) is an emerging paradigm that offers the execution of scientific workflows as a service. The service provider lies either in the Platform as a Service (PaaS) or Software as a Service (SaaS) layer based on the cloud service model. WaaS providers make use of distributed computational resources to serve the enormous need for computing power in the execution of scientific workflows. They provide a holistic service to scientists starting from the user interface in the submission portal, applications installation, and configuration, staging of input/output data, workflow scheduling, and resource provisioning. WaaS platforms are designed to process multiple workflows from different users. The workload is expected to arrive continuously, and workflows must be handled as soon as they arrive due to the QoS constraints defined by the users. Therefore, WaaS platforms must deal with a high level of complexity derived from their multi-tenant and dynamic nature, contrary to a traditional WMS that is commonly used for managing a single workflow execution.

Several variations of WaaS framework—which extend the WMS architecture—are found in literature such as the work by Wang et al. [21] describing a service-oriented architecture that separates the components into user management layer, scheduler, storage, and VM management. Meanwhile, a framework with a similar division that emphasizes on distributed storage is proposed by Esteves and Veiga [22]. Another architecture for multi-tenant scientific workflows execution in clouds emplaces the proposed framework as a service layer above the Infrastructure as a Service (IaaS) layers [23]. In general, we identified three primary layers in WaaS platforms: the tenant layer, the scheduler layer, and the resource layer. Based on the identified requirements of WaaS platforms, we propose an architecture for this system focusing on the scheduler component, as depicted in Figure 2.

First, the tenant layer manages the workflows submission, where users can configure their preferences and define the QoS of their workflows. The scheduler layer is responsible for placing the tasks on either existing or newly acquired resources and consists of four components: task runtime estimator, task scheduler, resource provisioner, and resource monitor. The task runtime estimator is used to predict the amount of time a task will take to complete a specific computational resource. Another component, the task scheduler, is used to map a task into a selected virtual machine for execution. Meanwhile, resource provisioner is used to acquiring and releasing virtual machines from third-party providers. The resource monitor is used to collect the resource consumption data of a task executed in a particular virtual machine. These data are stored in a monitoring database and are used by the task runtime estimator to build a model to estimate the task's runtime. The third-party infrastructure with which the platforms interact, fall into the resource layer.

We explore the underlying problem of scheduling multiple workflows in various distributed systems that have taken place for more than a decade with explicitly focusing on the multi-tenancy aspect of the problem related to the scheduling and resource provisioning. In this study, we expect to gain some knowledge to design and develop a multi-tenant WaaS platform that pretty much resembles the problem of multiple workflows scheduling in multi-tenant distributed computing systems.

4 TAXONOMY

The scope of this study is limited to the theoretical algorithms developed for multiple workflows scheduling that represent the problem in multi-tenant WaaS platforms. In this section, we describe

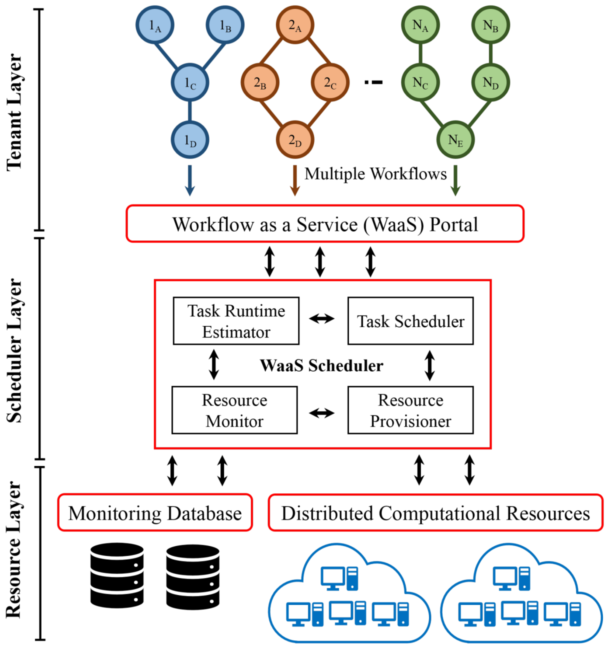

Fig. 2. Workflow as a service architecture.

various challenges of scheduling multiple workflows and their relevancy for each taxonomy classification. Furthermore, the mapping and references of the algorithms to each class are presented in Section 5.

4.1 Workload Model Taxonomy

Multiple workflows scheduling algorithms are designed to handle workloads with a high level of heterogeneity that represents a multi-tenant characteristic in the platforms. Workload heterogeneity can be described from several aspects, including the continuous arrival of workflows at different times, the various types of workflow applications that differ in computational requirements, the difference in workflow sizes, and the diversity in software libraries and dependencies.

The different arrival time of multiple workflows in the platforms resembles the problem of streaming data processing that deals with continuous incoming tasks to be processed. In contrast with some static single workflow scheduling algorithms that make use of information (e.g., workflow structure, the runtime of tasks, computational requirements) to create a near-optimal schedule plan, the continuous arrival of workflows in multi-tenant computing platforms makes this an unsuitable approach. Furthermore, conventional techniques to achieve near-optimal schedules such as metaheuristics are computationally intensive, and the complexity will grow as the workflow size increases. The time for planning may take longer than the actual workflow execution. Hence, a lightweight dynamic scheduling approach is the most suitable for multi-tenant environments as the algorithms must be able to deal with the dynamicity of the workload. For instance, at peak time, the concurrency of requests may be very high, whereas, at other times, the submission rate may reduce to a point where the inter-arrival time between workflows is long enough to execute each workflow in a dedicated set of resources.

The variety of application types is another issue to be addressed. A study by Juve et al. [24] shows a variety of workflow applications with different characteristics. The Montage astronomy workflow [25] that is used to reconstruct mosaics of the sky is considered a data-intensive application with high I/O activities. The CyberShake workflow [26] that is used to characterize earthquake hazards using the Probabilistic Seismic Hazard Analysis (PSHA) techniques is categorized as a compute-intensive workflow with multiple reads on the same input data. The Broadband workflow that is used to integrate a collection of simulation codes and calculations for earthquake engineers has a relative balance of CPU and I/O activities in its tasks. These three samples show different types of workflow applications that may have different strategies for scheduling to be carried out. For example, a strategy of clustering tasks with a high dependency of input/output data (i.e., data-intensive) and allocating the same resource for them to minimize data transfer.

Furthermore, heterogeneity is also related to the size of the workflows. The size represents the number of tasks in a workflow and may differ even between instances of the same type of workflow application due to different input datasets. For example, the Montage astronomy workflow takes the parameters of width and height degree of a mosaic of the sky as input. The higher the degree, the larger the workflow size to be executed as it resembles the size and shape of the sky area to be covered and the sensitivity of the mosaic to produce. A large-scale workflow may raise another issue in scheduling such as high volume data transfer that may cause a bottleneck in the network, which will affect other smaller scale workflows being executed in the platform.

Another heterogeneity issue is the various software libraries and dependencies required for different workflows. This problem is related to the deployment and configuration of workflows in the platforms. Deploying different software libraries and dependencies in a system requires technical efforts such as installing the software and managing conflicts between software dependencies. The most important implication related to this case is the resource sharing between workflows to utilize idle time slots produced during the scheduling. In cluster and grid environments where every user uses shared installed software systems on a physical machine, the conflicting dependencies are inevitable. This problem can be avoided by isolating applications in virtualized environments such as clouds.

However, in clouds, where the workflow's deployment and configuration can be isolated in a virtual machine, the possibility to share the computational power between users in a particular virtual machine is limited. This problem is due to the limitation of a virtual machine capacity (i.e., memory, storage) and possible conflicting dependencies if we want to have as much as software configured in a virtual machine instance. The tradeoff between the isolation and the resource sharing in clouds can be solved using container technology as successfully implemented using Kubernetes on Docker containers [27], Singularity and CVMFS at OSG [28], and Shifter at Blue Waters [29]. In this case, container, a lightweight operating system-level virtualization, is used to isolate the workflow application before deploying them on virtual machines. Therefore, both isolation and resource sharing objectives can be achieved. Based on the heterogeneity issue, these workloads can be differentiated by their workflow type and user-defined QoS requirements, as shown in Figure 3.

4.1.1 Workflow Type. Scheduling algorithms for multi-tenant computing platforms must consider the fact that the users in this system may submit a single type or different workflow applications. These variations can be categorized into homogeneous and heterogeneous workflow types. A homogeneous workflow type assumes all users submit the same kind of workflow applications (e.g., multi-tenant platforms for Montage workflow). In this case, the algorithms can be tailored to handle a specific workflow application by exploiting its characteristics (e.g., topological structure, computational requirements, software dependencies, and libraries). For example, related to a topological structure, a workflow with a large number of tasks in a level may raise an issue of data transfer. This issue can potentially become a communication bottleneck when all of the tasks in

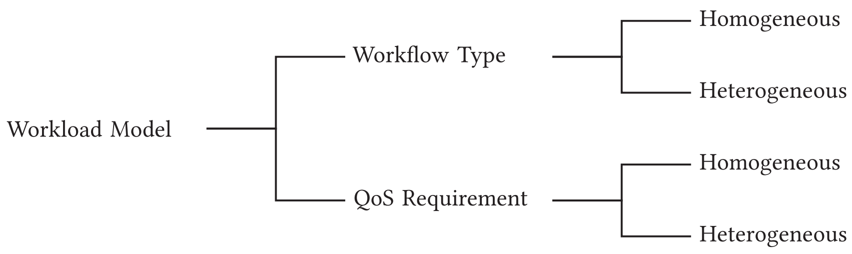

Fig. 3. Workload model taxonomy.

a level concurrently transfer the data input needed to execute the tasks. Therefore, clustering the tasks may result in a reduction in data transfer and eliminates the bottleneck in the system.

Furthermore, the heterogeneity from the resource management perspective affects how the scheduling algorithms handle software dependencies and libraries installed in computational resources. The algorithms for a homogeneous workflow type can safely assume that all resources contain the same software for a typical workflow application. In this way, the constraints for choosing appropriate resources for particular tasks related to the software dependencies can be eliminated, since all of the resources are installed and configured for the same workflow application.

However, to handle a heterogeneous workflow type, the algorithms must be able to tackle all various possibilities of workflow type submitted into the platforms. In a multi-tenant computing platform, where the heterogeneous workflow type is considered, tailoring the algorithms to the specific workflow application characteristics is impractical. The scheduling algorithms must be designed following a more general approach. For example, related to the topological structure, a workflow's task is considered ready for the execution when all of its predecessors are executed, and its data input is available in a resource allocated for execution. In this way, the algorithms can exploit a simple heuristic to build a scheduling queue by throwing in all tasks with this specification to the queue.

Therefore, a variety of software dependencies and libraries required for different workflow applications increases the possible conflict of software dependencies in platforms that consider heterogeneous workflow type. Therefore, the algorithms must include some rules in the resource selection step to determine what relevant resources can be allocated for specific tasks. For example, the algorithms can define a rule that is only allowing a task to be assigned a resource based on its software dependencies and libraries' availability.

4.1.2 QoS Requirements. Workloads in multi-tenant computing platforms must be able to accommodate multiple users' requirements. These requirements are represented by the QoS parameters defined when users submit their workflows to the platforms. We categorize the workloads based on the users' QoS requirements into homogeneous and heterogeneous QoS requirements.

The majority of algorithms designed for multi-tenant computing platforms surveyed in this study consider a homogeneous QoS requirement. They are designed to achieve the same scheduling objective (e.g., minimizing the makespan, meeting the budget) for all workflows. Meanwhile, a heterogeneous QoS requirement is addressed by the ability to be aware of various objectives and QoS parameters demanded by a particular user. The algorithms may consider several strategies within the same platforms to handle workflows with different QoS requirements. For example, to process workflows that are submitted with the deadline constraints, the algorithms may exploit the option to schedule them into the cheapest resources to minimize operational cost as long as their deadlines can be met. At the same time, the algorithms can also handle workflows with the budget constraints by using another option to lease as much as possible the fastest resources within the available budget.

4.2 Deployment Model Taxonomy

Handling the performance variability in multi-tenant distributed computing systems is essential to the multiple workflows scheduling problems, as the scheduling highly relies on the accurate estimation of workflow's performance on a particular computational infrastructure. Attempts to increase the quality of scheduling by accurately estimating the time needed for completing a task, as one of the strategies for taking care of the uncertainty, has been extensively studied [30]. Specific work designed for scientific workflows includes a work by Nadeem and Fahringer [31] that utilized the template to predict the scientific workflow application execution time. Another work by da Silva et al. [32] introduced an online approach to estimate the resource consumption for scientific workflows. Meanwhile, Pham et al. [33] worked on machine learning techniques to predict task runtime in workflows using a two-stage prediction approach.

When we specifically discuss cloud environments, the uncertainty becomes higher than cluster and grid environments. The virtualization that is the backbone of clouds is the primary source of the performance variability as reported by Leitner and Cito [11] and also previously discussed by Jackson et al. [34]. The cloud instances performance varies over time due to several aspects, including the virtualization overhead, the geographical location of the data center, and especially the multi-tenancy of clouds. For example, it is not uncommon for a task to have a longer execution time during a specific time (i.e., peak hours) in cloud instances due to the number of users served by a particular cloud provider at that time. The main conclusions from their works substantiate our assumption that the performance and predictability of clouds is something that is not easy to address.

Another variable of uncertainty in clouds is the provisioning and de-provisioning delays of VMs. When a user requests to launch a cloud's instance, there is a delay between the request and when the VM is ready to use, called provisioning delay. There also exists a delay in releasing the resource, namely de-provisioning delay. Not considering the provisioning and de-provisioning delays in the scheduling phase may cause a miscalculation of when to acquire and to release the VM. This error may cause an overcharging bill of the cloud services. A study by Mao and Humphrey [35] reported that the average provisioning delay of a VM, observed from three cloud providers—Amazon EC2, Windows Azure, and Rackspace—was 97 s while more recently, Jones et al. [36] presented a study that shows that three different cloud management frameworks—OpenStack, OpenNebula, and Eucalyptus—produced VM provisioning delays between 12 to 120 s. However, delays are not only derived from acquiring and releasing instances. As most of the WMS treat cloud instances (i.e., virtual machines) as virtual clusters using third-party tools (e.g., HTCondor[1]), there exists a delay in integrating a provisioned VM from cloud providers into a virtual cluster. An upper bound delay of 60 s for this process was observed by Murphy et al. [37] for an HTCondor virtual cluster. These delays are one of the sources of uncertainty in clouds, and therefore, the algorithms should consider then to get an accurate scheduling result.

Hence, the scheduling algorithms for multi-tenant computing platforms can be differentiated based on its deployment model. We identified two types of algorithms for multi-tenant computing platforms based on their deployment model, as illustrated in Figure 4. Several issues and challenges that arise from these deployment models are worthy of being considered by the scheduling algorithms.

[1]https://research.cs.wisc.edu/htcondor/.

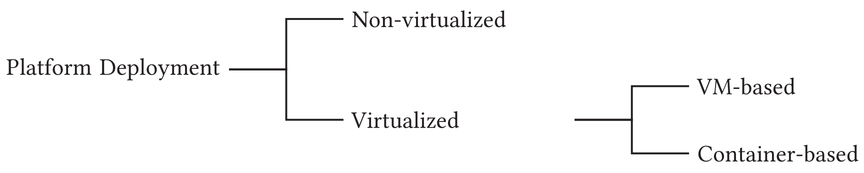

Fig. 4. Deployment model taxonomy.

4.2.1 Non-virtualized. The majority of works in our survey design are scheduling algorithms for cluster and grid environments. These two environments are the traditional way of establishing multi-tenant distributed computing systems where a large number of computational resources are connected through a fast network connection so that many users in a shared fashion can utilize it. However, in this way, there is no isolation between software installed related to their dependencies and libraries within the same physical machine.

Accommodating multi-tenant users in a non-virtualized environment is limited by the computational infrastructure static capacity. This staticity makes it very hard to auto-scale the resources in non-virtualized environments. Thus, the algorithms cannot efficiently serve a dynamic workload without having a queueing mechanism to schedule overloaded requests at a particular time. For example, adding a node into an established cluster infrastructure is possible but may involve technicalities that cannot be addressed in a short period. This environment also does not allow the users to shrink and expand their allocated resources quickly, since it needs to go through the administrator intermediaries. Therefore, the primary concern of scheduling algorithms designed for this environment is to ensure for maximum utilization of available resources, so the algorithms can reduce the queue of users waiting to execute their workflows. In this case, the techniques such as task rearrangement [38] and backfilling [39] can be used to fill the gaps produced by scheduling a particular workflow by allocating these idle slots to other workflows.

4.2.2 Virtualized. The algorithms designed for virtualized environments (i.e., clouds) can gain advantages from a flexible configuration of VM as it isolates specific software requirements needed by a user in a virtualized instance. A fully configured virtual machine can be used as a template and can be shared between multiple users to run the same workflows. This isolation ensures little disturbance to the platforms and the other users whenever a failure occurs. However, in this way, the possible computational sharing of a virtual machine is limited. It is not plausible to configure a virtual machine for several workflow applications at the same time. In this case, containers can be used to increase configuration flexibility in virtualized environments. The container is an operating-system-level virtualization method to run multiple isolated processes on a host using a single kernel. The container is initially a feature built for Linux that is further developed and branded as a stand-alone technology (e.g., Docker[2]) that not only it can run on Unix kernel but also on Windows NT kernel (e.g., windows container[3]). A full workflow configuration can be created in a container before deploying it on top of virtual machines. In this way, the computational capacity of VMs can be shared between users with different workflows.

In the context of scalability, algorithms designed for virtualized environments can comfortably accommodate multi-tenant requirements. The algorithms can acquire more resources in on-demand fashion whenever requests are overloading the system. Furthermore, this on-demand flexibility supported by the pay-as-you-go pricing scheme reduces the burden for the multi-tenant computing platform providers to make upfront payments for reserving a large number of resources

[2]https://www.docker.com/.
[3]https://docs.microsoft.com/en-us/virtualization/windowscontainers/.

that may only be needed at a specific time (i.e., peak hours). Even if a particular cloud provider cannot meet the demand of the multi-tenant computing platform providers, the algorithms can provision resources from different cloud providers.

However, this environment comes with a virtualization overhead that implies a significant performance variability. The overhead not only occurs from the communication bottleneck when a large number of users deal with high volume data but also the possible degradation of CPU performance, since the computational capacity is shared between several virtual machines in the form of virtual CPU. The other overheads are the delay in provisioning and de-provisioning virtual machines and the delay in initiating and deploying the container. The scheduling algorithms have to deal with these delays and consider them in the scheduling to ensure the accurate scheduling result.

4.3 Priority Assignment Model

Fairness and priority issues are unavoidable in multiple workflows scheduling. Given two workflows that arrive at the same time, the decision to execute a particular workflow must be determined using a policy to ensure that limited computational resources can be fairly allocated. This fairness can be identified from the *slowdown*, a difference of expected makespan between the execution of a workflow in dedicated resources vs. resource sharing environments [40]. In this case, the scheduling algorithms are designed to gain a slowdown value as low as possible. Since the computational resources are limited, the slowdown is inevitable. Therefore, the algorithms assigned a priority to the workflows to ensure the fulfillment of QoS. The priority can be derived from several aspects, including the QoS defined by users, the type of workflow application, the user's preference, and the size of the workflows.

Priority assignment can be determined based on the user-defined QoS. It is evident for scheduling algorithms to prioritize workflow with the earliest deadline, as this can ensure the fulfillment of QoS requirements. In this way, algorithms may introduce a policy based on the deadline that delays the scheduling of a workflow with a more relaxed deadline to improve resource sharing in the system without violating the fairness aspect. However, the priority assignment can be defined from the budget available. In real-world practice, it is common to prioritize the users with more budget available to do a particular job compared to the lower one (e.g., priority check-in for business class passengers). This policy also can be implemented in multiple workflows scheduling.

Assigning priority based on the type of application can be done by defining application or user classes. For example, workflows submitted for education or tutorial purpose may have a lower priority than the workflows executed in a scientific research project. Meanwhile, a workflow that is used to predict the typhoon occurrence may be performed first compared to a workflow for creating the mosaic of the sky. This policy can be defined out of the scheduling process based on some policies adopted by the providers.

Moreover, the priority assignment can also be determined based on the size of workflows. This approach is the most traditional way of priority scheduling that has been widely implemented, such as the Shortest Job First (SJF) policy, which prioritizes the smaller workflows over a larger one to avoid starvation. Another traditional scheduling algorithm like the Round-Robin (RR) also can be constructed based on the size of the workflows to ensure both of the small and large-scale workflows get fair treatment in the systems.

Fairness between users in multi-tenant computing platforms can be achieved through priority assignments. This assignment is essential as the ultimate goal of the providers is to fulfill each user's QoS requirement. We identify various priority assignment models from surveyed algorithms that consider the type of workflow application, users QoS constraints, user self-defined priority, and size of workflows in their design, as shown in Figure 5.

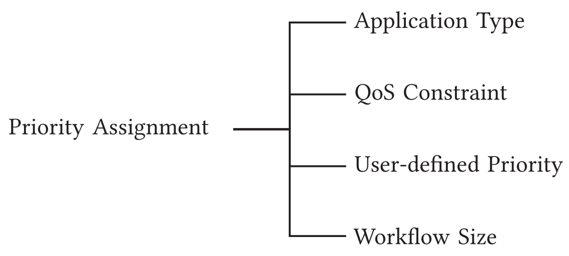

Fig. 5. Priority assignment model taxonomy.

4.3.1 Application Type. Different types of workflow applications can be used to define the scheduling priority based on their context and critical functionality. The same workflow application can differ in priority when it is used in a different environment. Montage workflow used for an educational purpose may have a lower priority than a solar research project using the same workflow. Meanwhile, considering the different critical functions of workflows and some events (e.g., earthquake), some workflow applications can be prioritized from the other. For example, CyberShake workflow to predict the ground motion after an earthquake may be prioritized compared to the Montage workflow that is used to create a mosaic of the sky image. This priority assignment needs to be designed in a specific policy of the providers that can regulate the fairness of the scheduling process.

4.3.2 QoS Constraint. Deriving priority assignments from users' QoS constraints can be done within the scheduling algorithms. This assignment is included in the logic of algorithms to achieve the scheduling objectives. For example, an algorithm that aims to minimize cost while meeting the deadline may consider to de-prioritize and delay the task of a particular workflow that has a more relaxed deadline to re-use the cheapest resources available. In this way, the algorithms must be designed to be aware of the QoS constraints of the tasks to derive these parameters into a priority assignment process during the scheduling.

Furthermore, the challenge of deriving priority assignment from QoS constraints may come from a heterogeneous QoS requirement workload. The algorithms must be able to determine a priority assignment for multiple workflows with different QoS requirements. For example, given two workflows with different QoS parameters, a workflow was submitted with a deadline, while another was included with a budget. The priority assignment can be done by combining these constraints with its application type, user-defined priority, or workflow structure.

4.3.3 User-defined Priority. On the contrary to the application type priority model that may be arranged through a specific policy, the priority assignment must also consider the user-defined priority in scheduling algorithms. This priority can be defined by users with appropriate compensations for the providers. For example, it is not uncommon in real-world practice to spend more money to get a prioritized treatment that affects the speed of process and quality of service (e.g., regular and express postal service). It is possible in multi-tenant computing platforms to accommodate such a mechanism where the users are given the option to negotiate their priority through a monetary cost compensation for the providers. This mechanism is a standard business practice adopted in multi-tenant computing platforms (e.g., reserved, on-demand, and spot instances pricing schemes).

4.3.4 Workflow Size. Another approach on priority assignment is based on the structure of workflows (e.g., size, parallel tasks, critical path). Prioritizing workflows based on their sizes resembles a traditional way of priority scheduling, such as SJF policy that gives priority to the shortest

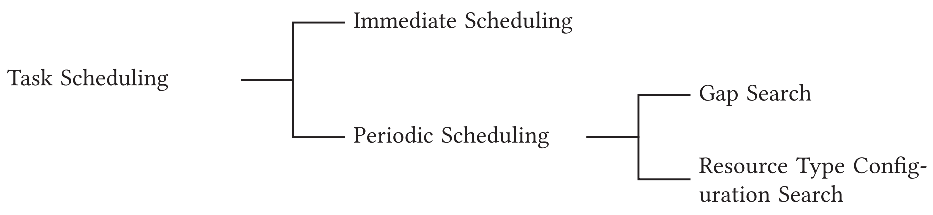

Fig. 6. Task scheduling model taxonomy.

tasks and RR policy that attempts to balance the fairness among tasks with different sizes. This prioritization can be combined with the QoS constraint to produce better fairness between users. For example, a large-scale workflow may have a very extended deadline. Therefore, smaller workflows with tight deadlines can be scheduled between the execution of tasks from this larger workflow.

4.4 Task Scheduling Model

Task-resource mapping is the primary activity of scheduling. All of the workflow scheduling algorithms have the purpose of finding the most optimal configuration of task-resource mapping. However, each scheduling problems may have different requirements regarding the QoS. In general, there are two standard QoS requirements in workflow scheduling, namely time and cost. The majority of the cases require the algorithms to minimize the overall execution time of the workflow (i.e., makespan).

However, the cost of executing the workflows significantly affects the scheduling decisions in utility-based computational infrastructures such as utility grids and cloud environments. Every user wants to minimize the cost of executing their workflows. These two objectives have opposing goals, and a tradeoff between them must be considered. This tradeoff then is derived into various scheduling objectives such as minimizing cost while meeting the deadline (i.e., time limit for execution), minimizing makespan while meeting the budget (i.e., cost limit for execution), or a more loose objective, meeting deadline, and budget.

In multi-tenant computing platforms, QoS diversity is prevalent due to the different needs of users to execute their workflows. The variety is not only related to the QoS values the users define but also may raise in the form of different scheduling objectives. The various user-defined QoS requirements must be handled in a way that each user's need can be fulfilled without sacrificing the other users served by the systems.

All algorithms in this study avoid metaheuristics approaches that are known for their complexity in planning the schedule before runtime. This planning creates an overhead waiting delay as the continuous arriving workflows have to wait for pre-processing before the actual scheduling takes place. Therefore, they use dynamic approaches that reduce the need for intensive computing at the planning phase and aim to achieve a fast scheduling decision by considering the current status of the systems. These approaches can be divided into immediate and periodic scheduling, as illustrated in Figure 6.

4.4.1 Immediate Scheduling. Immediate scheduling or just-in-time scheduling is a dynamic scheduling approach in which tasks are scheduled whenever they are ready for scheduling. In the case of multiple workflows, this scheduling approach collects all of the ready tasks from different workflows in a task pool before deciding to schedule based on some particular rules. The immediate scheduling tries to overcome the fast dynamic changes in the environments by adapting the decision based on the current status of the system. However, as the algorithm schedules the tasks based on a limited amount of information (i.e., a limited view of the previous and future information), this approach cannot achieve an optimal scheduling plan. However, it is an efficient way for multi-tenant platforms that deal with uncertain and dynamic environments.

The immediate scheduling resembles list-based heuristics scheduling. This scheduling approach, in general, has three scheduling phases: task prioritization, task selection, and resource selection. The algorithms repeatedly select a particular task from the scheduling queue that is constructed based on some prioritization method and then pick the appropriate resource for that specific task. For example, in deadline constraint-based heuristics algorithms that aim to minimize the cost while meeting the deadline, the scheduling queue is constructed based on the earliest deadline first (EDF) of the tasks and the cheapest resources that can meet the deadline are chosen to minimize the cost. The time complexity for heuristic algorithms is low. Therefore, it is suitable for multiple workflows scheduling algorithms that deal with the speed to manage the scheduling process.

4.4.2 Periodic Scheduling. This approach schedules the tasks periodically to exploit the possibility of optimizing a set of tasks' scheduling within a period. While in a general batch scheduling, a particular set is constructed based on the size of workload (i.e., schedule the tasks after reaching a certain number), the periodic approach schedules the tasks in a set of timeframe. In this case, the periodic scheduling acts as a hybrid approach between static and dynamic scheduling methods; static such that an algorithm exploits the information of a set of tasks (i.e., structures, estimated runtime) to create an optimal plan, but it do not need to wait for a full workload of tasks to be available. The dynamic sense of algorithms adapts and changes the schedule plan periodically. Hence, periodic scheduling refers to a scheduling technique that utilizes the schedule plan of a set of tasks available in a certain period to produce a better scheduling result. This method is more adaptable to changes and has faster pre-runtime computation than static scheduling techniques, since it only includes a small fraction of workload to be optimized rather than the entire workload. However, this approach can achieve a better result from having an optimized schedule plan than typical immediate scheduling but with less speed to schedule the tasks.

One of the approaches in periodic scheduling is identifying the gaps between tasks during the schedule. The identification uses an estimated runtime of tasks and their possible position in a resource during runtime. The most common methods to fill the gap are task rearrangement and backfilling strategy. Task rearrangement strategy rearranges the tasks scheduling plan to ensure the minimal gap in a schedule plan while backfilling allocates the ordering list of tasks and then backfills the holes produced between the allocation using the appropriate tasks. Both strategies do not involve an optimization algorithm that requires intensive computation, since the multi-tenant computing platforms consider speed in the schedule to cope with the users' QoS requirements.

While gap search is related to the strategy for improving resource utilization, another approach utilizes resource type configuration to optimize the cost spent on leasing computational resources. In a heterogeneous environment where resources are leased from third-party providers with some monetary costs (i.e., utility grids, clouds), determining resource type configuration to optimize the cost of leasing resources is necessary. For example, the Dyna algorithm [41], which considers the combination use of on-demand and spot instances in Amazon EC2, utilizes heuristics to find the optimal resource type configuration to minimize the cost.

4.5 Resource Provisioning Model

Resource provisioning forms an essential pair with task scheduling. In this stage, scheduling algorithms acquire and allocate resources to execute the scheduled tasks. We derive the categorization of resource provisioning based on the ability of scheduling algorithms to expand and shrink the number of resources within the platforms to accommodate the dynamic workloads of multi-tenant computing platforms, as shown in Figure 7.

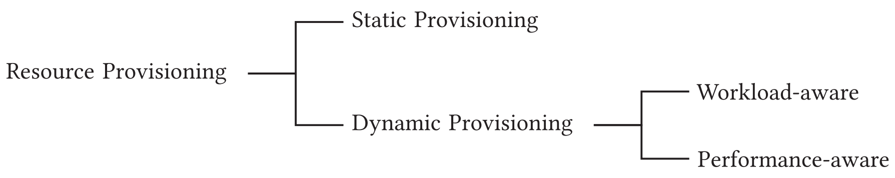

Fig. 7. Resource provisioning model taxonomy.

4.5.1 Static Provisioning. The static provisioning refers to scheduling algorithms where the number of resources used is relatively constant along the scheduling process. Therefore, the primary issue is related to the algorithms' ability to optimize the available resources to accommodate multiple users. This condition can be observed from the algorithms that emphasize heavily on the prioritization technique for workflows to be scheduled due to the limited available resources contested by many users. Another aspect is the improvement of resource utilization of the systems, which describes the ability of algorithms to allocate a limited number of resources efficiently.

This static provisioning is not exclusive to non-virtualized environments (e.g., clusters, grids), where the number of resources is hardly changing over time. This case also prevails in cloud computing environments where the providers determine the number of VMs to be leased before initiating the platforms, and the number remains unchanged over time. In this scenario, the scheduling algorithms do not consider any resource provisioning strategy to scale up and down resources when facing a dynamic workload of workflows.

4.5.2 Dynamic Provisioning. As the clouds provide elastic provisioning of virtual machines, the scheduling algorithms of multi-tenant computing platforms in clouds take advantage of the dynamic provisioning approach. The automated scaling of resources that can be easily implemented in clouds has been widely adopted mainly in the scheduling algorithms that deal with a dynamic workload. In this case, the need for resources can be high at a point (i.e., peak hours), while at the same time, the operational cost must be kept at a minimum. To minimize the cost of leasing VMs, they have to be released when the request is low. From the existing algorithms, there are two different approaches to auto-scaling the cloud instances, which are workload-aware and performance-aware.

Workload-aware dynamic provisioning is related to the ability of algorithms to become aware of the workload status in multi-tenant computing platforms and then to act according to the situation. Suppose that in the peak hours condition the algorithms acquire more VMs to accommodate the requests. One of the heuristics used in this scenario is based on the deadline constraints of the workload. Examples include when the algorithms use a task's deadline to decide whether a task should re-use available VMs, provision a new VM that can finish before the deadline, or delay the schedule to re-use future available VMs as long as it does not violate the deadline. This decision is essential, as the dynamic workload is common in multi-tenant computing platforms where the systems cannot predict the future status of the workload. Using this heuristic, provisioning additional VMs is more accurate, as the acquired new VM is based on the requirement of a particular task being scheduled.

However, the performance-aware dynamic provisioning refers to an approach of auto-scaling the VMs based on total resource utilization of current provisioned VMs. The algorithms monitor the status of the systems and acquire additional VMs when the usage is high and release several idle VMs when the utilization is low. Maintaining resource utilization at a certain threshold ensures the efficiency of multi-tenant computing platforms in the scheduling process. The majority of works considering this approach are the ones that consider only homogeneous VM types in their systems.

Table 2. Taxonomy of Workload and Deployment Model

Algorithms	Refs.	Keywords	Workload Model				Deployment Model		
			Workflow Type		QoS Requirements		Non-virtualized	Virtualized	
			Homogen	Heterogen	Homogen	Heterogen		VM-based	Container-based
RANK_HYBD	[42]	Dynamic-guided scheduling	-	✓	✓	-	✓	-	-
OWM	[48]	Scheduling framework	-	✓	-	✓	✓	-	-
MOWS	[39]		-	✓	-	✓	✓	-	-
P-HEFT	[46]	Dynamic-parallel scheduling	-	✓	✓	-	✓	-	-
MQMW	[44]	Multi-QoS scheduling	-	✓	✓	-	-	✓	-
MQSS	[45]		-	✓	✓	-	✓	-	-
EDF_BF	[51]	Exploiting schedule gaps	-	✓	✓	-	✓	-	-
EDF_BF_IC	[52]		-	✓	✓	-	✓	-	-
CWSA	[23]		-	✓	✓	-	-	✓	-
FSDP	[53]	Fairness & priority	-	✓	✓	-	✓	-	-
F_DMHSV	[54]		-	✓	✓	-	✓	-	-
FDWS	[40]		-	✓	✓	-	✓	-	-
Adaptive dual-criteria	[55]	Partition-based scheduling	-	✓	✓	-	✓	-	-
MLF_ID	[56]		-	✓	✓	-	✓	-	-
OPHC-TR	[57]	Privacy constraint	✓	-	✓	-	-	✓	-
Dyna	[41]	Deadline constraint	-	✓	✓	-	-	✓	-
EPSM	[58]		-	✓	✓	-	-	-	✓
DGR	[38]	Task rearrangement	-	✓	✓	-	✓	-	-
FASTER	[59]	Fault-tolerant	-	✓	✓	-	-	✓	-
EnReal	[60]	Energy-efficient	-	✓	✓	-	-	✓	-
EONS	[61]		-	✓	✓	-	-	✓	-
DPMMW & GESMW	[62]		-	✓	✓	-	-	✓	-
PRS	[63]	Uncertainty-aware	-	✓	✓	-	-	✓	-
EDPRS	[64]		-	✓	✓	-	-	✓	-
ROSA	[65]		-	✓	✓	-	-	✓	-
NOSF	[66]		-	✓	✓	-	-	✓	-
EDF_BF *In-Mem*	[67]	Data-locality perspective	-	✓	✓	-	✓	-	-
MW-HBDCS	[68]	Deadline-budget constraints	-	✓	✓	-	✓	-	-
MW-DBS	[49]		-	✓	✓	-	✓	-	-
MQ-PAS	[50]	Profit-aware	-	✓	✓	-	-	✓	-
CERSA	[69]	Real-time scheduling	-	✓	✓	-	-	✓	-

In this way, the algorithms do not need to perform the complicated selection process of the VM types to be chosen.

5 SURVEY

This section discusses a number of surveyed multiple workflows scheduling algorithms from 2008 to 2019 that are relevant to our scope. Each algorithm is classified based on the taxonomy presented in the previous section. The classification of existing algorithms is depicted in Tables 2 and 3. Furthermore, the experimental design and parameter settings of each algorithm are described separately in Appendix A.

Table 3. Taxonomy of Priority Assignment, Task Scheduling, and Resource Provisioning Model

Algorithms	Refs.	Priority Assignment Model				Task Scheduling Model			Resource Provisioning Model		
							Periodic			Dynamic	
		App. Type	QoS Const.	User-defined	Wf Structure	Immediate	Gap Search	Conf. Search	Static	Workload	Performance
RANK_HYBD	[42]	-	-	-	✓	✓	-	-	✓	-	-
OWM	[48]	-	-	-	✓	-	✓	-	✓	-	-
MOWS	[39]	-	-	-	✓	-	✓	-	✓	-	-
P-HEFT	[46]	-	-	-	✓	✓	-	-	✓	-	-
MQMW	[44]	-	✓	-	-	✓	-	-	✓	-	-
MQSS	[45]	-	✓	-	-	✓	-	-	✓	-	-
EDF_BF	[51]	-	✓	-	-	-	✓	-	✓	-	-
EDF_BF_IC	[52]	-	✓	-	-	-	✓	-	✓	-	-
CWSA	[23]	-	✓	-	-	-	✓	-	-	-	✓
FSDP	[53]	-	✓	-	-	✓	-	-	✓	-	-
F_DMHSV	[54]	-	-	-	✓	-	✓	-	✓	-	-
FDWS	[40]	-	-	-	✓	✓	-	-	✓	-	-
Adapt. dual-criteria	[55]	-	✓	-	✓	-	✓	-	✓	-	-
MLF_ID	[56]	-	✓	-	-	✓	-	-	-	✓	-
OPHC-TR	[57]	✓	✓	-	-	✓	-	-	-	✓	-
Dyna	[41]	-	✓	-	-	-	-	✓	-	✓	-
EPSM	[58]	-	✓	-	-	✓	-	-	-	✓	-
DGR	[38]	-	✓	-	-	-	✓	-	✓	-	-
FASTER	[59]	-	✓	-	-	-	✓	-	-	✓	-
EnReal	[60]	-	✓	-	-	-	-	✓	-	✓	-
EONS	[61]	-	-	-	-	✓	-	-	-	✓	-
DPMMW & GESMW	[62]	-	✓	-	-	-	✓	-	-	✓	-
PRS	[63]	-	✓	-	-	✓	-	-	-	✓	-
EDPRS	[64]	-	✓	-	-	✓	-	-	-	✓	-
ROSA	[65]	-	✓	-	-	✓	-	-	-	✓	-
NOSF	[66]	-	✓	-	✓	✓	-	-	-	✓	-
EDF_BF *In-Mem*	[67]	-	✓	-	-	-	✓	-	✓	-	-
MW-HBDCS	[68]	-	✓	-	✓	✓	-	-	✓	-	-
MW-DBS	[49]	-	✓	-	✓	✓	-	-	✓	-	-
MQ-PAS	[50]	-	✓	-	✓	✓	-	-	✓	-	-
CERSA	[69]	-	✓	✓	-	✓	-	-	-	✓	-

5.1 Planner-Guided Scheduling for Multiple Workflows

RANK_HYBD algorithm [42] was introduced to overcome the impracticality of the ensemble approach (i.e., merging multiple workflows) to handle different submission time of workflows to the system by scheduling individual tasks dynamically. The algorithm put together all ready tasks from different workflows into a pool. Then, the algorithm used a modified upward ranking [43], which calculates the weight of a task based on its relative position from the exit tasks and estimated computational length to assign individual tasks priorities. This task ranking time complexity was $O\,(T_w \,.\, P_s)$ for all tasks in a workflow T_w given a set of static processors P_s. In contrast with the original upward ranking implementation in HEFT algorithm that chose tasks with higher rank value, RANK_HYBD preferred tasks with the lowest rank in the pool, which created a time complexity of $O\,(T_r\,.\,P_s)$ for re-prioritizing all ready tasks T_r. In this case, HEFT preferred the tasks from later arriving workflows and the tasks with the most extended estimated runtime, which created an unfair pre-emptive policy for the running workflows. By using the opposite approach, the RANK_HYBD algorithm avoided the pre-emptive scheduling delay of a nearly finished workflow if a new workflow is submitted in the middle of the execution. Finally, it schedules each task to the processor that can give the earliest processing time with $O\,(T_r\,(T_r\,.\,P_s))$ time complexity.

In general, the time complexity was quadratic to the number of tasks processed. The report showed that RANK_HYBD outperformed RANDOM and FIFO algorithms by 1.77× average speedup on workloads up to 25 multiple workflows. This algorithm was the first solution for multiple workflows scheduling. Many later algorithms have adopted the approach to tackling the dynamic workload of workflows using dynamic prioritization for tasks within a workflow and between workflows. Even though many aspects, such as QoS constraints, performance variability, and real workflow applications, have not been included in the experiment, this pioneering work became an important benchmark for the following algorithms.

5.2 Multiple QoS Constrained Scheduling for Multiple Workflows

The Multiple QoS Constrained Scheduling Strategy of Multi-Workflows (MQMW) algorithm [44] incorporated a similar strategy of RANK_HYBD to schedule multiple workflows. MQMW prioritized tasks dynamically based on several parameters, including resource requirement of a task, time and cost variables, and covariance value between time and cost constraint. This task ranking time complexity was $O\,(T_w\,.\,C_s)$ for all tasks in a workflow T_w given a set of static cloud instances C_s. The algorithm preferred the tasks with a minimum requirement of resources to execute, minimum time and cost limit, and a task with minimum covariance between its time and cost limit (i.e., when time limit decreases, the cost will increase). Each time the scheduling takes place, MQMW re-compute all ready tasks T_r by $O\,(T_r\,.\,C_s)$ time complexity. Finally, MQMW schedules each task to the best fit idle cloud instances with $O\,(T_r(T_r\,.\,C_s)\,)$ time complexity.

In general, the time complexity was quadratic to the number of tasks processed. It is tested against RANK_HYBD, even though the RANK_HYBD was not considered the cost in the scheduling constraint. The evaluation results showed that MQMW outperformed the success rate of RANK_HYBD [42] algorithm by 22.7%. MQMW was the first attempt to provide the solution of multiple workflows scheduling on the cloud computing environment. However, its cloud model did not resemble the real characteristics that are inherent in clouds such as elastic scalability of instances, on-demand resources, pay-as-you-go pricing schemes, and performance variability of cloud environments.

MQSS algorithm [45] was proposed to overcome shortcomings from the previous MQMW algorithm that includes the number of QoS considered in the scheduling and the adoption of a more optimal scheduling strategy. With the relatively same approaches, the MQSS includes other QoS parameters into the scheduling's attributes (e.g., time, cost, availability, reputation, and data quality). In general, MQSS has the same time complexity as MQMW, with 12.47% success rate improvement for the same workloads.

5.3 Fairness in Multiple Workflows Scheduling

The Parallel Task HEFT (P-HEFT) algorithm [46] was the first work of a group from the Universidade do Porto that modeled the non-monotonic tasks (i.e., the execution time of a task might differ on the different number of resource usage). The algorithm used a relative length position of a task from the entry task (i.e., t-level/top-level) and exit task (i.e., b-level/bottom-level) to assign the priorities between tasks. This ranking time complexity was $O\,(T_w\,.\,P_s)$ for all tasks in a workflow T_w given a set of static processors P_s. In their case, the task model was different from other works as it allowed parallel execution of a task in several processors. Furthermore, the processor selection and task scheduling for all ready tasks T_r was $O(T_r\,(T_r\,.\,P_s))$. In general, the complexity was quadratic to the number of tasks processed. The evaluation results showed that P-HEFT outperformed the static algorithm HPTS by 1.52× average speedup on workloads of 12 multiple workflows.

The next work from this group was the Fairness Dynamic Workflow Scheduling (FDWS) algorithm [40]. FDWS chose a single ready task from each workflow into the pool instead of putting all ready tasks together. Local prioritization within a workflow utilized upward-rank mechanism while the task selection from different workflows to schedule used a percentage of the remaining task number of workflow the task belongs to (PRT) and a task position in its workflow's critical path (CPL). This prioritization time complexity was $O(T_w\,.\,P_s)$ for all tasks in a workflow T_w given a set of static processors P_s. Meanwhile, the resource selection and task scheduling for all ready tasks T_r in FDWS was $O(T_r\,(T_r\,.\,P_s))$. In general, the complexity was quadratic to the number of tasks processed. The evaluation results showed that FDWS outperformed RANK_HYBD and OWM by 1.1× and 1.15× average speedup, respectively, on workloads of 50 multiple workflows.

The Multi-Workflow Deadline-Budget Scheduling (MW-DBS) algorithm [49] was their work that addressed the utility aspect of a heterogeneous multi-tenant distributed system. This algorithm included deadline and budget as constraints. Furthermore, local priority was assigned using the same method from FDWS that creates $O(T_w\,.\,P_s)$ time complexity. However, instead of using PRT and CPL, MW-DBS used the task's deadline and workflow's scheduled tasks ratio for assigning global priority. Finally, MW-DBS modified the processor selection phase, in which it included a budget limit for task processing as a quality measure with $O(T_r\,.\,P_s)$ time complexity. Furthermore, the complexity of resource selection and task scheduling for all ready tasks T_r was $O(T_r\,(T_r\,.\,P_s))$. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that MW-DBS outperformed the success rate of FDWS [40] and its variants by 43% on workloads of 50 multiple real-world application workflows.

The latest work was the Multi-QoS Profit-Aware Scheduling (MQ-PAS) algorithm [50]. MQ-PAS was designed not only for the cloud computing environment but also for the general utility-based distributed system. Task ranking and selection complexity in MQ-PAS was $O(T_w\,.\,C_s)$ for all tasks in a workflow T_w given a set of static cloud instances C_s. Meanwhile, the quality measure in cloud instances selection was $O(T_r\,.\,C_s)$. Furthermore, the complexity of resource selection and task scheduling for all ready tasks T_r was $O(T_r\,(T_r\,.\,C_s))$. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that MQ-PAS outperformed the success rate of FDWS [40] by only 1% but significantly 20% improvement of profit on workloads of 50 multiple real-world application workflows.

Several variations of scheduling scenarios were covered in their works. One of the specific signatures from this group is the strategy of choosing a single ready task from workflow to compete in the scheduling cycle with the other workflows. This strategy represents the term "Fairness" that becomes the primary concern in most of their works. However, with their broad scenarios that were intended to cover the general process in a multi-tenant distributed computing systems, the different requirements in clouds from utility grids (e.g., billing period schemes, dynamic and uncertain environment) were not considered in their works.

5.4 Online Multiple Workflows Scheduling Framework

A group from the National Chiao-Tung University focused on developing a scheduling framework for multiple workflows scheduling. Their first algorithm, called Online Workflow Management (OWM) [48], consisted of four phases: critical path workflow Scheduling (CPWS), task scheduling, multi-processor task rearrangement, and adaptive allocation (AA). The CPWS phase ranked all tasks T_w based on their relative position in their workflows before they were submitted to the scheduling queue to create a schedule plan with $O(T_w \,.\, P_s)$ time complexity. If there were some gaps in a schedule plan, then task rearrangement took place to fill the gaps to improve resource utilization that creates $O({T_r}^2)$ time complexity. Furthermore, AA scheduled the highest priority task from the queue that was constructed based on the near-optimal schedule plan whose time complexity was $O(T_r\,(T_r \,.\, P_s + P_s)$. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that OWM outperformed RANK_HYBD by 1.15× average speedup on workloads of 100 multiple workflows.

In their following work, they extended OWM into the Mixed-Parallel Online Workflow Scheduling (MOWS) algorithm [39]. They modified the CPWS phase using the Shortest-Workflow-First (SWF) policy combined with the critical path prioritization. Then, MOWS used the priority-based backfilling to fill the hole of a schedule in the task rearrangement stage. The pre-emptive task scheduling policy was introduced in the AA phase, so the algorithm allowed the system to schedule the shortest workflow with a pre-emptive strategy and stopped it when the higher priority workflow was ready to run. The difference between MOWS and OWM is the task rearrangement phase, which used the priority-based backfilling that takes a lower time complexity of $O(T_r)$. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that MOWS outperformed OWM by 1.25× average speedup on workloads of 100 multiple workflows.

Both OWM and MOWS utilized periodic scheduling, in which periodically created a schedule plan for a set of ready tasks before submitting it to the scheduling queue. In this way, the algorithm can produce better scheduling results without having intensive computation beforehand. However, this approach may still create a bottleneck if the number of ready tasks in the pool increases. Implementing a strategy to create a fairness scenario when selecting ready tasks to reduce the complexity of calculating a schedule plan may work to enhance this scheduling framework.

5.5 Real-time Multiple Workflows Scheduling

One of the active groups that focused on real-time and uncertainty aspects of multiple workflows scheduling was the group from The Aristotle University of Thessaloniki, Greece. They did impressive works on multiple workflows scheduling that explicitly addressed the uncertainty in cloud computing environments.

Their first work was the Earliest Deadline First with Best Fit (EDF_BF) algorithm [51]. EDF policy was used for the task selection phase, and the BF was the strategy for exploiting the schedule gap. EDF_BF incorporated schedule gap exploitation that can be identified through the estimated position of a task's execution in a specified resource. From all of the possible positions, the algorithm exploited the holes using a bin packing technique to find the best fit for a task's potential position. The result can also be used to determine which resource should be selected for that specific task. Given a set of ready tasks T_r processed each time and static processor P_s available, task selection complexity in EDF_BF was $O(T_r \,.\, P_s)$. Meanwhile, the processor selection and schedule gap exploitation was $O(T_r\,(T_r + T_r \,.\, P_s)$. In general, the complexity was quadratic to the number of tasks processed. The evaluation results showed that EDF_BF outperformed the guarantee ratio (i.e., success rate) of its variants with Highest-Level First (HLF) and Least-Space-Time First (LSFT) policy on task selection by an average of 10%.

Another work was the Earliest Deadline First with Best Fit and Imprecise Computation (EDF_BF_IC) algorithm [52], which extended the previous algorithm with imprecise computation. The imprecise computation was first introduced in Reference [70] to tackle the problem in a real-time environment that was often needed to produce an early proximate result within a specified time limit. The imprecise computation model was implemented by dividing the task's components into a mandatory and optional component. A task was considered meeting the deadline if its mandatory part was completed, while the optional component may be fully executed, partially executed, or skipped. The evaluation results showed that EDF_BF_IC outperformed the success rate of its baseline EDF by an average of 16% and cost-saving improvement by 12%.

Furthermore, this group explored data-locality and in-memory processing for multiple workflow scheduling [67]. In this case, they combined the EDF_BF algorithm with a distributed in-memory storage solution called Hercules [71] to evaluate a different way of communication of workflow tasks. They considered two different communication scenarios, communication through a network, and via temporary files utilizing the Hercules in-memory storage solution. The results showed that scheduling performance increased when the I/O to computation ratio reduced by using in-memory storage, which enforced the locality of data. The evaluation results showed that the application completion ratio (i.e., success rate) improves as the tardiness bound (i.e., soft deadline ratio) increases while the average makespan deteriorates. Besides, the average makespan of the completed workflows improves as the I/O activities decrease.

Despite the variation, their algorithm's main idea was to schedule all of the ready tasks using EDF policy for resources that can allow the tasks to finish at their earliest time. The algorithm maintained a local queue for each resource and then optimized the local allocated queue using gaps filling techniques and, in one of the works, manipulated a small portion of the tasks that may have a little significance (i.e., imprecise computation). Their algorithms were designed for a multi-tenant system with a static number of resources. Therefore, the design may not be suitable for cloud computing environments—which suffer most from uncertainty problems—where the auto-scaling of resources is possible.

5.6 Adaptive and Privacy-aware Multiple Workflows Scheduling

A group from The University of Sydney introduced an excellent work of multiple workflows scheduling that was concerned with the privacy of users [57]. They developed two algorithms: Online Multiterminal Cut for Privacy in Hybrid Clouds using PCP Ranking (OMPHC-PCPR) and Online Scheduling for Privacy in Hybrid Clouds using Task ranking (OPHC-TR). OMPHC-PCPR was merging multiple workflows into one single workflow before scheduling. Hence, this solution is out of our scope, but the other one, OPHC-TR, used an approach that is inclusive of our study. Both algorithms calculated the privacy level of each workflow before they decided to schedule them in private or public clouds. The private clouds were used mainly for the workflow that comprised a high level of privacy parameters. The main differences between the two algorithms were their input. While OMPHC-PCPR considered a merged single workflow from several workflows, OPHC-TR processed each task using a rank mechanism to decide which tasks were submitted into the scheduling queue. Given a set of tasks in a workflow T_w and static processors P_s available, task ranking and selection complexity in OPHC-TR was $O\,(T_w\,.\,P_s)$. Meanwhile, the resource selection and task scheduling for all ready tasks T_r was $O\,(T_r\,(T_r\,.\,P_s))$. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that OMPHC-PCPR outperformed the cost-saving of the OPHC-TR algorithm by 50%. However, in this work, the overhead of merging several workflows in OMPHC-PCPR did not being evaluated thoroughly. Such an approach may result in a very high bottleneck when the number of workflows arriving reached a certain high particular number.

Another work from this group was the DGR algorithm [38]. This algorithm used heuristics, which started the solution with the initial reservation of resources for particular scheduling tasks that time complexity was $O\,(T_w\,.\,P_s)$ for all tasks in a workflow T_w given a set of static processors P_s. During the execution, uncertainty (i.e., performance and execution time variation) may profoundly affect the initial reservation and break the schedule plan. In this case, the algorithm rescheduled the tasks to handle the broken reservation. DGR utilized task rearrangement techniques and exploited a dynamic search tree to fix this reservation with $O(T_r(P_s + T_r . P_s))$ time complexity. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that DGR outperformed a traditional HEFT algorithm by 1.43× average speedup on workloads of 300 multiple workflows.

5.7 Adaptive Dual-criteria Multiple Workflows Scheduling

Another adaptive approach in scheduling multiple workflows was an adaptive dual-criteria algorithm [55]. This algorithm used heuristics that utilized scheduling adjustment via task rearrangement. An essential strategy to this algorithm was the clustering of tasks and treated them as an integrated set in scheduling to minimize the critical data movement within tasks. Hence, any rearrangement or adjustment to fill the schedule holes involved the set of tasks to be moved. Given a set of tasks in a workflow T_w processed each time, task group after clustering T_g where $T_g \leq T_w$, and static processors P_s available, the task initial clustering process complexity was $O({T_w}^2)$. Meanwhile, the adjustment of idle time gap selection was $O(T_g(T_g + P_s))$; it was ranked with higher complexity compared to a simple Best-Fit and EFT calculation of a single task due to the T_g constraint. Finally, the complexity of adaptive task group re-arrangement was $O(T_g P_s)$. In general, the time complexity was quadratic to the number of tasks and task groups processed. The evaluation results showed that this algorithm outperformed a similar process using traditional Best-Fit and EFT approaches by up to 1.41× speedup in various scenarios.

Since the approach used was the periodic scheduling, the frequency of scheduling cycle becomes critical. The infrequent scheduling cycle implies to the broader set of tasks to be processed, which may result in a more optimized scheduling plan but potentially required a more intensive computation for creating the plan. Meanwhile, a perpetual cycle may fasten the scheduling plan computation due to its size of tasks but may reduce the quality of a schedule. This variation was not being addressed and explored in-depth by the authors. Besides, the treatment of a cluster of tasks increases the coarse-granularity of scheduling that may widen the gaps produced. In this way, the task rearrangement may hardly find the holes that can be fit by a coarse-grained set of clustered tasks.

5.8 Multiple Workflows Scheduling on Hybrid Clouds

Another work designed for hybrid cloud environments was the Minimum-Load-Longest-App-First with the Indirect Transfer Choice (MLF_ID) algorithm [56]. The term Load-Longest-App had a similar concept to the critical path. So MLF_ID was a heuristic algorithm that incorporated the workflows prioritization based on their critical path and exploited the use of private clouds before leasing the resources in public clouds. MLF_ID partitioned the workflow based on a hierarchical iterative application partition (HIAP) to eliminate data dependencies between a set of tasks by clustering tasks with dependencies into the same set before scheduling them into either private or public clouds.

Given a set of tasks in a workflow T_w processed each time, static private cloud resources C_s, and dynamic public cloud resources C_d, the application partition complexity was $O(T_w(C_s + C_d))$. Meanwhile, the ready tasks T_r scheduling that included the decision to schedule on public cloud was $O(T_r . C_d)$ or private cloud was $O(T_r(T_r + C_s))$. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that the combined resources of hybrid clouds could minimize the total execution cost when the number of workflows can be allocated as much as possible to the private resources. However, the private cloud capacity was restrained, as the scaling process is not as simple as such an approach in public clouds.

The use of hybrid clouds in this work was emphasized to extend the computational capacity when the available on-premises infrastructure (i.e., private clouds) were not enough to serve the workloads. First, the tasks were scheduled to the private clouds, and whenever it was not possible to process the capacity, they were transferred to public clouds. Even though the tasks had been partitioned to make sure that the data transfer between them was minimal, the decision to move to public clouds evoked a possible transfer overhead problem. Therefore, some improvements can be made by implementing a policy to decide whether a set of tasks is considered impractical to process in private clouds because they include some algorithm to predict the possible future overhead in the system. In this way, instead of directly transferring the execution to the public clouds, which results in not only additional cost but also transfer overhead, the algorithm can decide whether it should move the execution or delay the process and wait for the next available resources.

5.9 Proactive and Reactive Scheduling for Multiple Workflows

Another group that focused on real-time and uncertainty problems in scheduling was a group from The National University of Defense Technology, China. They proposed algorithms that dynamically exploited proactive and reactive methods in multiple workflows scheduling.

Their first work was the Proactive Reactive Scheduling (PRS) algorithm [63]. The proactive phase calculated the estimated earliest start and execution times of tasks and then scheduled them dynamically based on a simple list-based heuristic. This method had been incorporated into many algorithms for multiple workflows scheduling. However, using only the proactive method results in an inability to tackle the uncertainties (e.g., performance variation, overhead delays), which led to sudden changes in the system. Then, PRS introduced a reactive phase whenever two disruptive events occurred (i.e., the arrival of new workflow and finishing time of a task). The reactive phase was triggered by two disruption events to re-do (i.e., update) the scheduling process based on the latest system status. Given a set of tasks in a workflow T_w processed each time and dynamic cloud resources C_d available, the time complexity of task ranking was $O(T_w)$. Meanwhile, the VM selection and task scheduling for all ready tasks T_r was $O(T_r(T_r . C_d)$. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that PRS outperformed the cost-savings of modified SHEFT [72] and RTC [73] algorithms for multiple workflows by 50.94% and 67.23%, respectively, on workloads of 1,000 multiple workflows.

They then extended PRS into Event-driven and the Periodic Rolling Strategies (EDPRS) algorithm [64]. EDPRS tackled a flaw in PRS whereby if none of the two disruption events happened, the scheduling process could not be pushed forward. They introduced a periodic rolling strategy (i.e., scheduling cycle) that drove the re-iteration of the schedule. In this way, when no disruption events occur, the algorithm repeats their scheduling activities after a specific periodic rolling time. In general, the time complexity is similar to that of the PRS algorithm. The evaluation results showed that EDPRS outperformed the cost-savings of modified SHEFT [72] and RTC [73] algorithms for multiple workflows by 12.98% and 21.57%, respectively. Both PRS and EDPRS worked well in handling the uncertainty in cloud computing environments.

This group also worked on energy-efficient multiple workflow scheduling algorithms. They produced the Energy-Efficient Online Scheduling (EONS) algorithm [61]. EONS was different from the other energy-efficient scheduling algorithms due to its focus on fast and real-time-oriented scheduling. EONS utilized simple auto-scaling techniques to lower energy consumption instead of optimizing energy usage using techniques such as VM live migration and VM consolidation. The scaling method used simple heuristics that considered the load of the physical host and the hardware efficiency. Given a set of tasks in a workflow T_w processed each time and dynamic cloud resources C_d available, task ranking complexity in EONS was $O(T_w)$. Meanwhile, the VM selection and task scheduling for all ready tasks T_r was $O(T_r(T_r . C_d)$. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that EONS outperformed the energy savings of modified EASA [74] and ESFS [75] algorithms for multiple workflows by 45.64% and 35.98%, respectively.

Another work from this group addressed the failure in multiple workflows scheduling. The algorithm, called FASTER [59], utilized the primary backup technique to handle the failure. To the best of our knowledge, this was the only fault-tolerant algorithm for multiple workflows scheduling. As part of the pre-processing phase, they scheduled two copies of a task (i.e., primary and backup copy) based on the FCFS policy. The workflows were accepted for execution when both primary and backup copies successfully met their deadlines in the estimation phase. Whenever a task was not able to meet its deadline, the algorithm re-calculates its earliest start time. This estimation takes $O(T_w{}^2)$ given a set of tasks in a workflow T_w processed each time. FASTER then ensured that the primary copy was distributed among all available hosts as part of its fault-tolerant strategy. This heuristic requires periodic scanning of all VMs C_d within the available physical host H_d in the system. The complexity of host monitoring phase was $O(H_d(C_d \,.\, T_w))$ for each primary and backup type of tasks. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that FASTER outperformed the modified eFRD [76] algorithm for multiple workflows by 239.66% in terms of guarantee ratio (i.e., success rate) and 63.79% in terms of resource utilization.

Their next algorithms were called ROSA [65] and CERSA [69]. These algorithms were the improvement of PRS and EDPRS algorithms that specifically tackle the uncertainties in executing multiple real-time workflows. While their previous algorithm EDPRS relies on a periodic trigger to clear a task pool beside the arrival of new workflows, ROSA and CERSA initiate the scheduling based on specific disturbance events. ROSA defined triggering events like the arrival of new workflows and the completion of a task in a particular cloud instance. However, CERSA added the arrival of the urgent task as one of the triggering events. In general, the time complexity of CERSA and ROSA is quadratic, similarly to the PRS and EDPRS. The evaluation results showed that in terms of monetary cost, ROSA outperformed EPSM [58] and CWSA [23] by 10.07% and 23.18%, while CERSA outperformed CWSA [23] and OPHC-TR [57] by 8.31% and 17.22%, respectively.

The algorithms emphasize a specific strategy to handle real-time scenarios by using an immediate scheduling approach, which includes the update strategy to adapt to changes dynamically. However, this dynamic approach, especially on the energy-efficient and fault-tolerant problem, can be improved by optimizing the VM placement on the physical machine, since the algorithms may have access to the information of the physical infrastructure.

5.10 Energy Aware Scheduling for Multiple Workflows

A group from Nanjing University, China, proposed an algorithm for multiple workflows scheduling that was called EnReal, an energy-aware resource allocation method for workflow in the cloud environment [60]. While the previous energy-aware algorithm (EONS) utilized auto-scaling techniques to lower the energy consumption, EnReal exploited the VM live migration-based policy. The algorithm partitioned all of the ready tasks in the scheduling queue based on their requested start time and allocated them to the resources on the same physical machine. The adjustment was made whenever a load of the physical machine was exceeding the threshold, and then VM live migration policy took place.

EnReal also adjusted the VM allocation dynamically whenever a task was finished. Combined with the physical machine resource monitoring, the global resource allocation method emphasized the energy saving of the platform. However, its partitioning method did not consider the dependencies between the tasks that imply a data transfer overhead when they were allocated to different physical machines. The energy-aware resource allocation policy in EnReal should have complemented by an ability to aware of data-locality. This policy will not only minimize energy consumption but also improves the scheduling results in terms of total execution cost and makespan. In general, the most intensive phase was the resource monitoring that takes quadratic time complexity. Furthermore, the performance evaluation results showed that EnReal outperformed the modified energy-aware Greedy-D [77] algorithm in terms of energy efficiency by 18% on average.

5.11 Monetary Cost Optimization for Multiple Workflows on Commercial Clouds

A group from the National University of Singapore proposed Dyna [41], an algorithm that concerned with the clouds' dynamicity nature. They introduced a probabilistic guarantee of any defined SLAs of workflow users as it was the closest assumption to the uncertainty environment in clouds. This approach was a novel contribution, since the majority of the works assumed deterministic SLAs in their algorithms. Dyna aimed to minimize multiple workflows scheduling execution cost by utilizing VMs with spot instances pricing scheme in Amazon EC2 along with its on-demand instances. Dyna started with the initial configuration of different cloud instance types and refined the configuration iteratively to get the better scenario that minimizes the cost while meeting the deadline. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that Dyna outperformed the monetary cost of the modified MOHEFT algorithm [78] by 74% on average.

Dyna presented an exploration of possible cost reduction in executing multiple workflows by utilizing the spot instances in Amazon EC2. Since multi-tenant computing platforms that were assumed in their work acted as a service provider for many users, the use of reserved instances in Amazon EC2 may further reduce the cost of running the platform. Comparison among on-demand, spot, and reserved instances in Amazon EC2 needs to be done further to deepen the plausible scenario on minimizing the execution cost of multiple workflows in clouds.

5.12 Fairness Scheduling for Multiple Workflows

Fairness Scheduling with Dynamic Priority for Multi Workflow (FSDP) [53] was an algorithm proposed by a group from Dalian University of Technology, China. FSDP emphasized the fairness aspect as it incorporated slowdown metrics into their algorithm's policy. The slowdown value is the ratio of the makespan of a workflow when it is being scheduled in dedicated service to the makespan of it being scheduled in a shared environment with the other workflows. The closer the slowdown value is to 1, the fairest the algorithm scheduled the workflows in the system. FSDP also included an urgency metric, a value that represented the priority of each workflow based on its deadline. The slowdown and urgency were updated periodically when a workflow finished ensuring the refinement in the scheduling process.

However, the fairness scenario was not explored in-depth by the authors. FSDP is only evaluated using two different workflows on a various number of resources (i.e., processor). The issue of fairness would arise when the number of submitted workflows was high enough to represent the condition of peak hour in multi-tenant distributed computing systems. In general, the time complexity was quadratic to the number of tasks processed. The evaluation results showed that FSDP slightly outperformed the overall makespan of MMHS algorithm [79].

5.13 Scheduling Tradeoff of Dynamic Multiple Workflows

A group from Hunan University presented two algorithms. The first one was the Fairness-based Dynamic Multiple Heterogeneous Selection Value (F_DMHSV) algorithm [54]. The algorithm consisted of six steps, which were task prioritization, task selection, task allocation, task scheduling, new workflow arrival handling, and task monitoring. Task prioritization used a descending order of heterogeneous priority rank value (HPRV) [80], which included the out-degree (i.e., number of successors) of the task. This prioritization complexity was $O(T_w \,.\, P_s)$ for all tasks in a workflow T_w given a set of static processors P_s. The task was selected from the ready tasks pool based on the maximum HPRV. Furthermore, the task was allocated to the processor with minimum heterogeneous selection value (HSV) [80], which optimized the task allocation criteria using the combination of upward and downward rank, which created a complexity of $O(T_r \,.\, P_s)$ for all ready tasks T_r. The task, then, was scheduled to the earliest available processor with minimum HSV. The evaluation results showed that F_DMHSV outperformed RANK_HYBD, OWM, and FDWS algorithms by 1.37×, 1.11×, and 1.03× average speedup, respectively.

In the same year, this group published energy-efficient algorithms that combined the Deadline-driven Processor Merging for Multiple Workflow (DPMMW) algorithm that aimed to meet the deadline, and the Global Energy Saving for Multiple Workflows (GESMW) algorithm sought to lower energy consumption [62]. DPMMW was a clustering algorithm that allocated the clustered tasks in a minimum number of processors so that the algorithm can put idle processors into sleep mode. Meanwhile, GESMW re-assigned and adjusted the tasks to any processor with minimum energy consumption in the global scope. The combination of DPMMW and GESMW was exploited to get lower energy consumption. This approach was different from the previous two energy-efficient algorithms that focused on virtual machine level manipulation. In general, the most intensive phase in this algorithm was the invoking of the HEFT algorithm to create a baseline scheduling plan and traversing all processors, which take quadratic time complexity. Furthermore, the performance evaluation results showed that DPMMW&GESMW outperformed the energy saving of the reusable DEWTS, a modified version of the DEWTS algorithm [81] by 8.1% on average.

This group presented two opposite approaches to scheduling with different objectives. However, in both methods, the works emphasize a similar strategy of resource selection. In their first work, the algorithm focuses on selecting various resources to minimize the makespan, while in the second one, it is choosing the different machines with various energy efficiency to reduce energy consumption. These resource selection strategies can improve the scheduling result by combining them with efficient task scheduling approaches.

5.14 Workflow Scheduling in Multi-tenant Clouds

Another algorithm for multiple workflows scheduling was Cloud-based Workflow Scheduling (CWSA) [23]. This work used the term "multi-tenant clouds" in its paper for describing the multi-tenancy aspect that was generally considered in cloud computing environments, whereas the definition itself was similar to the multiple workflows we used in this survey. The algorithm was intended for compute-intensive workflows applications. Hence, CWSA ignored data-related overhead and focused on compute resource management. The algorithm aimed to minimize the total makespan of the workflows, which in the result, decreases the cost of execution. In general, the time complexity of CWSA is $O(T_w \,.\, C_d)$, given a set of workflow tasks T_w and a number of dynamic cloud instances C_d. The performance evaluation results showed that the CWSA outperformed both the makespan and cost of standard FCFS, EASY Backfilling, and Minimum Completion Time (MCT) policy.

However, CWSA did not further optimize its cost minimization strategy using a cost-aware resource provisioning technique. CWSA auto-scaled the resources using a resource utilization threshold, in which it acquired and released the resources if their utilization exceeded or was below a specific number. For example, they implemented the rule; if the usage was ≥70% for 10 minutes, then it was scaled-up by adding 1 VM of small size. In this case, the algorithms with cost-aware auto-scaling strategy—that specifically acquires and releases particular VMs based on the workload—may outperform CWSA that only considers overall system utilization based auto-scaling. This type of auto-scaling strategy is not provisioning resources that are specifically tailored to the need of workloads.

5.15 Multi-tenant Workflow as a Service Platform

The latest solution for multiple workflow scheduling was Elastic Resource Provisioning and Scheduling Algorithm for Multiple Workflows designed for Workflow as a Service Platforms (EPSM) [58]. This work used a specific term of "multi-tenant workflow as a service" to describe the platform for executing multiple workflows in the clouds. However, the "multi-tenant workflow as a service" term and "multiple workflows" can be used interchangeably in this case. The EPSM introduced a scheduling algorithm for multi-tenant computing platforms that utilized a container to bundle workflow's application before deploying it into VMs. In this way, the users can share the same VMs without having any problem related to software dependencies and libraries.

The algorithm consisted of two-phase resource provisioning that included a flexible approach of scaling up and down the resources to cope up with the dynamic workload of workflows and scheduling that exploited a delay policy based on the task's deadline to re-use the cheapest resources as much as possible to minimize the cost. In the resource provisioning phase, EPSM incorporated an overhead detection in the form of provisioning delay and de-provisioning delay of the VMs. This strategy was able to reduce unnecessary costs due to violating a coarse-grain billing period of clouds. The algorithm made an update of the unscheduled tasks' deadline whenever a task finished the execution. In this way, the algorithm dynamically adapted the gap between the estimated and actual execution plans to ensure scheduling objectives. In the scheduling phase, EPSM considered re-using available VMs before provisioning the new one to minimize the delay of acquiring new VMs and possible cost minimization by re-using the cheapest VMs available. In general, the time complexity of the EPSM algorithm is quadratic to the number of tasks processed. Furthermore, the performance evaluation results showed that this algorithm outperformed the cost-saving of the Dyna algorithm [41] by 19% on average for various scenarios.

5.16 Concurrent Multiple Workflows Scheduling

The latest work on deadline- and budget-constrained multiple workflows scheduling is the Multi-workflow Heterogeneous budget-deadline-constrained Scheduling (MW-HBDCS) algorithm [68], introduced by a group from Guangzhou University, China. This work used the term "concurrent multiple workflows" as it emphasized the concurrent condition of multiple workflows, which means tackling several workflows that arrived at the same time or overlapped on a dense condition. MW-HBDCS was designed to improve the flaw of the previous similar algorithm, MW-DBS [49]. Significant enhancement was the inclusion of a budget in the ranking process to prioritize the tasks for scheduling. The algorithm was also designed to tackle uncertainties in the environments. In this work, the authors used the terms "consistent" and "inconsistent" environments to describe various dynamicity in multi-tenant distributed computing systems. In general, the time complexity was quadratic, similarly to MWDBS time complexity. The evaluation results showed that MW-HBDCS outperformed the success rate of MW-DBS by 46% and 52% on synthetic and real-world workflow applications, respectively.

MW-HBDCS tackled many flaws that were not considered in the previous deadline- and budget-constrained scheduling algorithms. These enhancements were the model that incorporated high uncertainties and dynamicity, the improvement of a task's ranking mechanism that made the budget one of the primary constraints in addition to the deadline, which previously only acted as a complementary constraint. Since the authors considered the budget as crucial as the deadline, it is essential to include the tradeoff analysis between the values of the budget and deadline, as it is related to the success rate of workflows execution. One of the techniques to such an approach is the

Pareto analysis, which is used for multi-objective workflow scheduling (e.g., MOHEFT [78]). Furthermore, this algorithm considers static resource provisioning. Therefore, it may not achieve optimal performance in cloud computing environments where the auto-scaling of resources is possible.

5.17 Scheduling Multiple Workflows under Uncertain Task Execution Time

The latest deadline-aware multiple workflows scheduling algorithm is NOSF [66]. This algorithm adopted a similar strategy to several previous algorithms (e.g., EDPRS, EPSM, ROSA) designed to tackle the uncertainties in cloud computing environments. The NOSF aims to minimize the cost of leasing cloud instances by optimizing resource utilization using the sharing strategy of the VM billing period. To further distribute a fair share of sub-deadlines between tasks, NOSF made use of PCP to create end-to-end scheduling of several tasks during the deadline distribution process. Therefore, traversing workflows for detecting the PCP was the most intensive phase that takes quadratic time complexity.

NOSF relies on the strategy of maintaining minimal growth of the leased cloud instances instead of auto-scaling the resources dynamically by eliminating future idle VMs. This strategy may cause a problem of waiting overhead in very dense workflows' arrival (i.e., high concurrent workflows). Combining dynamic auto-scaling and maintaining low growth of VM leased may become an essential strategy for multi-tenant computing platforms with a quite high uncertainties situation. Furthermore, the performance evaluation results showed that NOSF outperformed ROSA [65] in terms of reducing the cost and deadline violation probability by an average of 50.5% and 55.7%, respectively, while improving resource utilization by 32.6% on average.

6 FUTURE DIRECTIONS

Many challenges and problems in the current solutions should be considered for the future of scientific workflows, as nicely discussed in a study by Deelman et al. [82]. However, this article describes the particular scheduling aspect explicitly in multi-tenant distributed computing systems. We capture the future direction of multi-tenancy from existing solutions and rising technologies that have a high potential to support the enhancement of multiple workflows scheduling. The range is broad from the pre-processing phase, which involves the strategy to accurately estimate task execution time that is a prerequisite for scheduling process; scheduling techniques that are aware of constraints such as failure, deadline, budget, energy usage, privacy, and security; and the use of heterogeneous distributed systems that differ not only in capacity but also pricing scheme and provisional procedures. We also observe a potential use of several technologies to enhance the multi-tenancy that comes from rising technologies such as containers, serverless computing, and Unikernels and the broad adoption of the Internet of Things (IoT) workflows.

6.1 Advanced Multi-tenancy Using Microservices

Microservice is a variant of service-oriented architecture (SOA) that has a unique lightweight or even simple protocol and treats the application as a collection of loosely coupled services [83]. In this sense, we can consider container technology, serverless computing (i.e., function as a service), and Unikernels to fall into this category.

Kozhirbayev and Sinnott [84] report that the performance of a container on a bare-metal machine is comparable to a native environment, since no significant overhead is produced during the runtime. It is a promising technology to enhance multi-tenancy features for multiple workflows scheduling, as it can be used as an isolated environment for workflow application before deploying it into virtual machines in clouds. We argue that, in the future, this technology will be widely used for solving the multi-tenancy problem, as it has been explored for executing a single scientific workflow as reported in several studies [85–88].

However, the main tradeoff of general-purpose container's performance (i.e., Docker) for scientific applications is security [89]. The multi-tenancy requirements inevitably invite multiple users sharing the same computational infrastructure at a time. In the case of Docker, every container process has access to the Docker daemon, which is spawned as a child of the root. At any rate, this activity compromised the whole IT infrastructure. To tackle this security problem, a Singularity Container that targets explicitly scientific applications has been developed [90]. It has been tested in the Comet Supercomputer at the San Diego Supercomputer Center [91] and has shown a promising result for handling multi-tenancy in the future workflow as a service environment. A study by Suhartanto et al. [92] shows that a container can exploit multiple molecular docking processes within a host without significant performance degradation. In this way, the algorithms should be able to enhance the multi-tenancy of a host VM by deploying multiple jobs with acceptable deterioration. However, the algorithms must consider the additional container initiating delay [93] as part of their design.

Another promising technology is serverless computing/Function as a Service (FaaS). FaaS is a new terminology that stands on the top of cloud computing as a simplified version of the virtualization service. In this way, cloud providers directly manage resource allocation, and the users only needed to pay for the resource usage based on the application codes. This technology facilitates users who need to run specific tasks from a piece of code without needing to manage the cloud instances. We will consider this as part of future directions, since there is high potential of its multi-tenancy service to accommodate multi-tenant workflows scheduling.

Furthermore, this technology has been tested for single scientific workflow execution, as reported by Malawski [94], Jiang et al. [95], and Malawski et al. [96]. Notably, this FaaS can serve workloads that consist of platform-independent workflows, which can be efficiently executed on top of this facility without having to provision a new virtual machine. Nevertheless, deploying scientific workflows in the serverless infrastructure is limited by the size of applications' requirements in terms of CPU, memory, and network, as reported by Spillner et al. [97].

Finally, Unikernels is another virtualization technology that is designed to maintain perfect isolation of virtual machines and maintain the lightweight of the container [98]. Unikernels enhance virtualization in terms of weight by removing a general-purpose OS from the VM. In this way, Unikernels directly run the application on the virtual hardware. Even more impressive, a recent finding shows that Unikernels do not require a virtual hardware abstraction. It can directly run as a process by using an existing kernel system, called a whitelisting mechanism [99]. Looking into the combination features of virtual machines and containers in one single virtualization technology, we can hope for a better multi-tenancy for the WaaS platform using this technology.

6.2 Reserved vs. On-demand vs. Spot Instances

The further reduction of operational cost has been a long-existing issue in utility-based multi-tenant distributed computing systems. Notably, in cloud computing environments where the resources are leased from third-party providers based on various pricing schemes, a cost-aware scheduling strategy is highly considered. Most of the algorithms for clouds use on-demand instances that ensure reliability in a pay-as-you-go pricing model. Meanwhile, Zhou et al. [41] explored the use of spot instances, which are cheaper but less reliable, as they have limited time available and could be terminated at an unpredictable time by the providers. This type of resource raises the issue of fault-tolerance.

However, the use of reserved instances in clouds should be explored to minimize further the total operational cost as the pricing of this bulk reservation is lower than on-demand even spot instances. The issue of using reserved instances is related to how accurate the algorithms can predict the workload of workflows to lease a number of reserved instances. The combination of reserved, on-demand, and spot instances must be explored to create an efficient resource provisioning strategy. The workload pattern forecasting model also should accompany the algorithms to better provision the cloud resources.

6.3 Multi-clouds vs. Hybrid Clouds vs. Bare-Metal Clouds

The use of multi-cloud providers for scientific workflow was explored by Jrad et al. [100] and Montes et al. [101] by introducing algorithms that were aware of different services available. However, the only relevant works found in our study are the use of hybrid clouds for separating tasks execution. In our survey, a work used hybrid clouds to treat tasks with different privacy levels in healthcare services, while another research utilized public clouds to cover the computational need that could not be fulfilled using private clouds and on-premises infrastructure. In our opinion, further utilization of multi-clouds can be beneficial, as a single cloud provider may not be able to serve the high requirements of resources in multi-tenant computing platforms on particular peak hours. The other advantage of multi-clouds is the reduction of operational cost as various cloud providers charge different prices for the datacenter in different geographical locations. In this way, discovering relevant services can be further explored to minimize data movements by choosing a particular data center location and also comparing the ratio of price and performance from various cloud instances from multiple cloud providers.

In this heterogeneity discussion, we have to mention a valuable service that provides a more heterogeneous infrastructure, called bare-metal clouds. Bare-metal cloud is an emerging service in the IaaS business model that leases a physical machine instead of a virtual machine to the users. This service targets users who need specific hardware requirements in an intensive computation (i.e., GPU, FPGA). While one may compare the elasticity of this service with any standard cloud services, recent work has shown that such agility in provisioning bare-metal clouds can be compared to general VM virtualization [102]. However, the challenge of managing such an environment must be considered when designing the algorithms. In this way, the scheduling policy should calculate the several possible overhead factors (i.e., network bandwidth, end-to-end latency) in comparison to the monetary cost of the infrastructure.

6.4 Fast and Reliable Task Runtime Estimation in Near-real-time Processing

Predicting task runtime in clouds is non-trivial, mainly due to the problem in which cloud resources are subject to performance variability [34]. This variability occurs due to several factors—including virtualization overhead, multi-tenancy, geographical distribution, and temporal aspects [11]—that affect not only computational performance but also the communication network used to transfer input/output data [103]. The majority of algorithms rely on the estimation of task execution time to produce an accurate schedule plan. Meanwhile, the works on task runtime estimation in scientific workflows are limited, including the latest works by Pham et al. [33] and Nadeem et al. [104] that used machine learning techniques. Previously, work on scientific workflow profiling and characterization by Juve et al. [24] that produced a synthetic workflow generator is being used by the majority of studies on workflow scheduling.

Future techniques must be able to address dynamic workloads that are continuously arriving, as in stream data processing. The adoption of an online and incremental machine learning approach may become another solution. In this approach, the algorithm does not need to learn from a model constructed from a large number of collected datasets, which is generally time-consuming and compute intensive. The algorithm only sees the data once and then integrates additional information as the model is incrementally built from new data. The latest work by Sahoo et al. [105] develops OMKR, an online and incremental machine learning approach, to handle large time-series datasets in a near real-time process. While this approach is still intensively being studied for scientific workflows, the preliminary work has been presented by Hilman et al. [106] for future workflows as a service platform. These methods should mainly consider the concept drift that impacted the performance variability of the clouds as reported for other continuous data stream problems [107].

6.5 Integrated Anomaly Detection and Fault-tolerant-aware Platforms

Detecting anomalies in scientific workflows is one of the challenges to maintain the fault-tolerance of scheduling in multi-tenant distributed computing systems. Several notable works are presented by Samak et al. [108] that detailed integrated workflows and resource monitoring for the STAMPEDE project. Furthermore, Gaikwad et al. [109] used Autoregression techniques to detect the anomalies by monitoring the systems, and a similar work by Rodriguez et al. [110] adopted Neural Network methods. However, the fault-tolerant algorithms found in our survey used replication technique [59] and checkpointing [41] to handle failure in workflows execution.

Future works in this area include the integration of detecting anomalies and failure-aware scheduling in multi-tenant computing platforms and the use of various fault-tolerant methods in failure-aware algorithms, such as resubmission and live migration. Furthermore, to investigate how the anomalies detection model can be combined with the task runtime prediction model to better schedule multiple tasks on heterogeneous environments. In this case, the algorithms should incorporate the ability to fully aware of the underlying hardware performance and their monitoring features. The algorithms then can decide to either resubmit the anomalous jobs or duplicated the jobs in the first place to ensure the completion of the workflows.

6.6 Multi-objective Constraints Scheduling

The flexibility and ability to easily scale the number of resources (i.e., VMs) in the cloud computing environment leads to a tradeoff between two conflicting QoS requirements: time and cost. In this case, the more powerful VMs capable of processing a task faster will be more expensive than the slower, less powerful ones. There has been an extensive research [10] on this scheduling topic, specifically designed for cloud computing environments, with most works proposing algorithms that were aimed at minimizing the total execution cost while finishing the workflow execution before a user-defined deadline. Meanwhile, the works that aimed to minimize the makespan by fully utilizing the available budget to lease as much as possible the faster resources are limited. We identified algorithms that considered budget in their scheduling, such as References [50, 68, 111, 112], which exploited the workflow budget as a complementary constraint to the deadline. Nevertheless, none of them aims to fully utilize the available budget to get a faster execution time.

Furthermore, studies on multiple workflows scheduling that is specifically aim to achieve multi-objective optimization (i.e., time and cost minimization) is also minimal. While metaheuristics and evolutionary programming have been used, such as in Fard et al. [113], the implementation for multiple workflows scheduling is limited by its pre-processing requirement. However, a more lightweight list-based heuristic approach, such as MOHEFT [78] and DPDS [114], can be considered for multiple workflows scheduling. In this case, the algorithms should carefully handle the tradeoff between achieving two or more objective constraints and maintaining the lightweight low-complexity scheduling process. Lightweight scheduling can be achieved by exploiting heuristics approaches to gain a relatively good optimization instead of aiming for the optimal schedule through sophisticated approaches with high computational cost, such as evolutionary and bio-inspired computing algorithms.

6.7 Energy-efficient Computing

Beloglazov et al. [115] have extensively explored the issue of green computing in cloud datacenters. Interestingly, there are several works in our study that addressed this energy-efficient and carbon footprint issue. While a work discussed energy-efficient strategy at the infrastructure level by implementing a live migration technique [60], another work tackled the problem at the workload level by allocating the load to specific physical machines [61].

For the multi-tenant computing platform providers that rely on IaaS clouds to lease the computational resources, adopting workload-level strategies for energy-aware scheduling is one of the possible further explorations. In this case, they do not have direct control over raw computational infrastructures as IaaS cloud providers do. Therefore, the algorithms should consider the energy-aware strategy of choosing the green computational resources, as discussed by Toosi et al. [116] in a work that explored a renewable-aware geographical load balancing.

6.8 Privacy-aware Scheduling

The users' privacy is an essential aspect that has been tackled by separating the execution in a private and public cloud-based on their data privacy level [57]. However, it is crucial to consider the security aspect of managing privacy, since both issues are highly inter-related. One of the works that consider security is the SABA algorithm [117]. However, it is designed for a single workflow scheduling and intended to explore the relationship between cost and security aspects in the scheduling, instead of focusing on the privacy aspect.

Further exploration of privacy and security in the multiple workflows scheduling has to be done as it resembles real-world workflow application problems. Another way to deal with privacy is by adopting a reliable security protocol for data processing in clouds, such as homomorphic encryption [118]. However, one inevitable tradeoff from the attempt to increase the security aspect is a delay to the total computational time. Multiple workflows scheduling algorithms should consider this tradeoff and include it as part of the scheduling strategy design.

6.9 Internet of Things Workflows

A vision paper by Gubbi et al. [119] mentions a future use of the Internet of Things (IoT) in a workflow form. The idea has been implemented in several works, including a smart city system [120] and a big data framework [121]. This type of workflow increases in numbers, and its broad adoption is predicted to occur in the near future. Therefore, the need for a multi-tenant computing platform that can handle such workflows may arise.

IoT applications are highly demanding network resources to handle end-to-end services from sensors to users. Therefore, network-intensive strategies such as bandwidth-aware and latency-aware must be considered in scheduling algorithms. A recent study by Stavrinides and Karatza [122] presents a work that is aware of edge and cloud resources available to differentiate task allocation based on their computational requirements.

7 SUMMARY

This article presents a study on algorithms for multiple workflows scheduling in multi-tenant distributed computing systems. In particular, the research focuses on the heterogeneity of workloads, a model for deploying multiple workflows, a priority assignment model for multiple users, scheduling techniques for multiple workflows, and resource provisioning strategies in multi-tenant distributed computing systems. It presents a taxonomy covering the focus of the study based on a comprehensive review of multiple workflows scheduling algorithms. The taxonomy is accompanied by classification from surveyed algorithms to show the existing solutions for multiple workflows scheduling in various aspects. The current algorithms within the scope of the study are reviewed and classified to open up the problems in this area and provide the readers with a helicopter view of multiple workflows scheduling. Some descriptions and discussions of various solutions are covered in this article to give a more detailed and comprehensive understanding of the state-of-the-art techniques and even to get an insight into further research and development in this area.

REFERENCES

[1] Roger Barga and Dennis Gannon. 2007. *Scientific versus Business Workflows*. Springer, London, 9–16. DOI: http://dx.doi.org/10.1007/978-1-84628-757-2_2

[2] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. 2014. *Workflows for e-Science: Scientific Workflows for Grids*. Springer.

[3] Carlos Goncalves, Luis Assuncao, and Jose C. Cunha. 2012. Data analytics in the cloud with flexible MapReduce workflows. In *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science*. 427–434. DOI: http://dx.doi.org/10.1109/CloudCom.2012.6427527

[4] Jia Yu and Rajkumar Buyya. 2005. A taxonomy of workflow management systems for grid computing. *J. Grid Comput.* 3, 3 (2005), 171–200. DOI: http://dx.doi.org/10.1007/s10723-005-9010-8

[5] Marek Wieczorek, Andreas Hoheisel, and Radu Prodan. 2008. *Taxonomies of the Multi-criteria Grid Workflow Scheduling Problem*. Springer US, 237–264. DOI: http://dx.doi.org/10.1007/978-0-387-78446-5_16

[6] Fuhui Wu, Qingbo Wu, and Yusong Tan. 2015. Workflow scheduling in cloud: A survey. *J. Supercomput.* 71, 9 (2015), 3373–3418. DOI: http://dx.doi.org/10.1007/s11227-015-1438-4

[7] Sukhpal Singh and Inderveer Chana. 2016. A survey on resource scheduling in cloud computing: Issues and challenges. *J. Grid Comput.* 14, 2 (2016), 217–264. DOI: http://dx.doi.org/10.1007/s10723-015-9359-2

[8] Ehab N. Alkhanak, Sai P. Lee, and Saif U. R. Khan. 2015. Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities. *Fut. Gener. Comput. Syst.* 50, Suppl. C (2015), 3–21. DOI: http://dx.doi.org/10.1016/j.future.2015.01.007

[9] Sucha Smanchat and Kanchana Viriyapant. 2015. Taxonomies of workflow scheduling problem and techniques in the cloud. *Fut. Gener. Comput. Syst.* 52, Suppl. C (2015), 1–12. DOI: http://dx.doi.org/10.1016/j.future.2015.04.019

[10] Maria A. Rodriguez and Rajkumar Buyya. 2017. A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. *Concurr. Comput.: Pract. Exp.* 29, 8 (2017), e4041–n/a. DOI: http://dx.doi.org/10.1002/cpe.4041

[11] Philipp Leitner and Jürgen Cito. 2016. Patterns in the Chaos: A study of performance variation and predictability in public IaaS clouds. *ACM Trans. Internet Technol.* 16, 3 (2016), 1–23. DOI: http://dx.doi.org/10.1145/2885497

[12] Saima G. Ahmad, Chee S. Liew, Muhammad M. Rafique, Ehsan U. Munir, and Samee U. Khan. 2014. Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems. In *Proceedings of the 4th IEEE International Conference on Big Data and Cloud Computing*. 129–136. DOI: http://dx.doi.org/10.1109/BDCloud.2014.63

[13] Kathy Svitil. 2016. Gravitational waves detected 100 years after Einstein's prediction. (2016). Retrieved from http://www.caltech.edu/news/gravitational-waves-detected-100-years-after-einstein-s-prediction-49777.

[14] Jun Qin and Thomas Fahringer. 2012. *Scientific Workflows: Programming, Optimization, and Synthesis with ASKALON and AWDL*. Springer Science & Business Media.

[15] Maria A. Rodriguez and Rajkumar Buyya. 2017. Scientific workflow management system for clouds. In *Software Architecture for Big Data and the Cloud*, Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, and Bruce Maxim (Eds.). Morgan Kaufmann, Boston, 367–387. DOI: http://dx.doi.org/10.1016/B978-0-12-805467-3.00018-1

[16] Bartosz Balis. 2016. HyperFlow: A model of computation, programming approach and enactment engine for complex distributed workflows. *Fut. Gener. Comput. Syst.* 55 (2016), 147–162. DOI: http://dx.doi.org/10.1016/j.future.2015.08.015

[17] Prakashan Korambath, Jianwu Wang, Ankur Kumar, Lorin Hochstein, Brian Schott, Robert Graybill, Michael Baldea, and Jim Davis. 2014. Deploying kepler workflows as services on a cloud infrastructure for smart manufacturing. *Proc. Comput. Sci.* 29 (2014), 2254–2259. DOI: http://dx.doi.org/10.1016/j.procs.2014.05.210

[18] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny. 2019. The evolution of the pegasus workflow management software. *Comput. Sci. Eng.* 21, 4 (2019), 22–36.

[19] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, and Carole Goble. 2013. The Taverna workflow

suite: Designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Res.* 41, 1 (2013), 557–561. DOI: http://dx.doi.org/10.1093/nar/gkt328

[20] Bill Howe, Garret Cole, Emad Souroush, Paraschos Koutris, Alicia Key, Nodira Khoussainova, and Leilani Battle. 2011. Database-as-a-service for long-tail science. In *Scientific and Statistical Database Management.* Springer, Berlin, 480–489.

[21] Jianwu Wang, Prakashan Korambath, Ilkay Altintas, Jim Davis, and Daniel Crawl. 2014. Workflow as a service in the cloud: Architecture and scheduling algorithms. *Proced. Comput. Sci.* 29, Suppl. C (2014), 546–556. DOI: http://dx.doi.org/10.1016/j.procs.2014.05.049

[22] Sérgio Esteves and Luís Veiga. 2016. WaaS: Workflow-as-a-service for the cloud with scheduling of continuous and data-intensive workflows. *Comput. J.* 59, 3 (2016), 371–383. DOI: http://dx.doi.org/10.1093/comjnl/bxu158

[23] Bhaskar P. Rimal and Martin Maier. 2017. Workflow scheduling in multi-tenant cloud computing environments. *IEEE Trans. Parallel Distrib. Syst.* 28, 1 (2017), 290–304. DOI: http://dx.doi.org/10.1109/TPDS.2016.2556668

[24] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. 2013. Characterizing and profiling scientific workflows. *Fut. Gener. Comput. Syst.* 29, 3 (2013), 682–692. DOI: http://dx.doi.org/10.1016/j.future.2012.08.015

[25] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. 2008. The cost of doing science on the cloud: The montage example. In *Proceedings of the ACM/IEEE Conference on Supercomputing.* 1–12. DOI: http://dx.doi.org/10.1109/SC.2008.5217932

[26] Philip Maechling, Ewa Deelman, Li Zhao, Robert Graves, Gaurang Mehta, Nitin Gupta, John Mehringer, Carl Kesselman, Scott Callaghan, David Okaya, Hunter Francoeur, Vipin Gupta, Yifeng Cui, Karan Vahi, Thomas Jordan, and Edward Field. 2007. *SCEC CyberShake Workflows—Automating Probabilistic Seismic Hazard Analysis Calculations.* Springer, London, 143–163. DOI: http://dx.doi.org/10.1007/978-1-84628-757-2_10

[27] Phuong Nguyen and Klara Nahrstedt. 2017. MONAD: Self-adaptive micro-service infrastructure for heterogeneous scientific workflows. In *Proceedings of the 2017 IEEE International Conference on Autonomic Computing.* 187–196. DOI: http://dx.doi.org/10.1109/ICAC.2017.38

[28] Lincoln Bryant, Jeremy Van, Benedikt Riedel, Robert W. Gardner, Jose C. Bejar, John Hover, Ben Tovar, Kenyi Hurtado, and Douglas Thain. 2018. VC3: A virtual cluster service for community computation. In *Proceedings of the Practice and Experience on Advanced Research Computing.* 30:1–30:8. DOI: http://dx.doi.org/10.1145/3219104.3219125

[29] Maxim Belkin, Roland Haas, Galen W. Arnold, Hon W. Leong, Eliu A. Huerta, David Lesny, and Mark Neubauer. 2018. Container solutions for HPC systems: A case study of using shifters on blue waters. In *Proceedings of Practice and Experience in Advanced Research Computing.* 1–8. DOI: https://doi.org/10.1145/3219104.3219145

[30] Carl Witt, Marc Bux, Wladislaw Gusew, and Ulf Leser. 2019. Predictive performance modeling for distributed batch processing using black box monitoring and machine learning. *Inf. Syst.* 82 (2019), 33–52. DOI: http://dx.doi.org/10.1016/j.is.2019.01.006

[31] Farrukh Nadeem and Thomas Fahringer. 2009. using templates to predict execution time of scientific workflow applications in the grid. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid.* 316–323. DOI: http://dx.doi.org/10.1109/CCGRID.2009.77

[32] Rafael F. da Silva, Gideon Juve, Mats Rynge, Ewa Deelman, and Miron Livny. 2015. Online task resource consumption prediction for scientific workflows. *Parallel Process. Lett.* 25, 3 (2015), 1541003. DOI: http://dx.doi.org/10.1142/S0129626415410030

[33] Thanh P. Pham, Juan J. Durillo, and Thomas Fahringer. 2017. Predicting workflow task execution time in the cloud using a two-stage machine learning approach. *IEEE Trans. Cloud Comput.* 99 (2017), 1–1. DOI: http://dx.doi.org/10.1109/TCC.2017.2732344

[34] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, and Nicholas J. Wright. 2010. Performance analysis of high performance computing applications on the amazon web services cloud. In *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science.* 159–168. DOI: http://dx.doi.org/10.1109/CloudCom.2010.69

[35] Ming Mao and Marty Humphrey. 2012. A performance study on the VM startup time in the cloud. In *Proceedings of the 5th IEEE International Conference on Cloud Computing.* 423–430. DOI: http://dx.doi.org/10.1109/CLOUD.2012.103

[36] Mike Jones, Bill Arcand, Bill Bergeron, David Bestor, Chansup Byun, Lauren Milechin, Vijay Gadepally, Matt Hubbell, Jeremy Kepner, Pete Michaleas, Julie Mullen, Andy Prout, Tony Rosa, Siddharth Samsi, Charles Yee, and Albert Reuther. 2016. Scalability of VM provisioning systems. In *Proceedings of the IEEE High Performance Extreme Computing Conference.* 1–5. DOI: http://dx.doi.org/10.1109/HPEC.2016.7761629

[37] Michael A. Murphy, Brandon Kagey, Michael Fenn, and Sebastien Goasguen. 2009. Dynamic provisioning of virtual organization clusters. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid.* 364–371. DOI: http://dx.doi.org/10.1109/CCGRID.2009.37

[38] Wei Chen, Young C. Lee, Alan Fekete, and Albert Y. Zomaya. 2015. Adaptive multiple-workflow scheduling with task rearrangement. *J. Supercomput.* 71, 4 (2015), 1297–1317. DOI: http://dx.doi.org/10.1007/s11227-014-1361-0

[39] Yirong Wang, Kuochan Huang, and Fengjian Wang. 2016. Scheduling online mixed-parallel workflows of rigid tasks in heterogeneous multi-cluster environments. *Fut. Gener. Comput. Syst.* 60, Suppl. C (2016), 35–47. DOI: http://dx.doi.org/10.1016/j.future.2016.01.013

[40] Hamid Arabnejad, Jorge G. Barbosa, and Frédéric Suter. 2014. Fair resource sharing for dynamic scheduling of workflows on heterogeneous systems. In *High-Performance Computing on Complex Environments*.

[41] Amelia C. Zhou, Bingsheng He, and Cheng Liu. 2016. Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds. *IEEE Trans. Cloud Comput.* 4, 1 (2016), 34–48. DOI: http://dx.doi.org/10.1109/TCC.2015.2404807

[42] Zhifeng Yu and Weisong Shi. 2008. A planner-guided scheduling strategy for multiple workflow applications. In *Proceedings of the International Conference on Parallel Processing*. 1–8. DOI: http://dx.doi.org/10.1109/ICPP-W.2008.10

[43] Haluk Topcuoglu, Salim Hariri, and Min You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* 13, 3 (2002), 260–274. DOI: http://dx.doi.org/10.1109/71.993206

[44] Meng Xu, Lizhen Cui, Haiyang Wang, and Yanbing Bi. 2009. A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications*. 629–634. DOI: http://dx.doi.org/10.1109/ISPA.2009.95

[45] Cui Lizhen, Xu Meng, and Yanbing Bi. 2009. A scheduling strategy for multiple QoS constrained grid workflows. In *Proceedings of the Joint Conferences on Pervasive Computing*. 561–566. DOI: http://dx.doi.org/10.1109/JCPC.2009.5420119

[46] Jorge G. Barbosa and Belmiro Moreira. 2011. Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters. *Parallel Comput.* 37, 8 (2011), 428–438. DOI: http://dx.doi.org/10.1016/j.parco.2010.12.004

[47] Jorge G. Barbosa, Celeste Morais, Ruben Nobrega, and Antònio Monteiro. 2005. Static scheduling of dependent parallel tasks on heterogeneous clusters. In *Proceedings of the IEEE International Conference on Cluster Computing*. 1–8. DOI: http://dx.doi.org/10.1109/CLUSTR.2005.347024

[48] Chihchiang Hsu, Kuochan Huang, and Fengjian Wang. 2011. Online scheduling of workflow applications in grid environments. *Fut. Gener. Comput. Syst.* 27, 6 (2011), 860–870. DOI: http://dx.doi.org/10.1016/j.future.2010.10.015

[49] Hamid Arabnejad and Jorge G. Barbosa. 2017. Maximizing the completion rate of concurrent scientific applications under time and budget constraints. *J. Comput. Sci.* 23, Suppl. C (2017), 120–129. DOI: http://dx.doi.org/10.1016/j.jocs.2016.10.013

[50] Hamid Arabnejad and Jorge G. Barbosa. 2017. Multi-QoS constrained and profit-aware scheduling approach for concurrent workflows on heterogeneous systems. *Fut. Gener. Comput. Syst.* 68, Suppl. C (2017), 211–221. DOI: http://dx.doi.org/10.1016/j.future.2016.10.003

[51] Georgios L. Stavrinides and Helen D. Karatza. 2011. Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. *Simul. Model. Pract. Theory* 19, 1 (2011), 540–552. DOI: http://dx.doi.org/10.1016/j.simpat.2010.08.010

[52] Georgios L. Stavrinides and Helen D. Karatza. 2015. A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds. In *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud*. 231–239. DOI: http://dx.doi.org/10.1109/FiCloud.2015.93

[53] Yuxin Wang, Shijie Cao, Guan Wang, Zhen Feng, Chi Zhang, and He Guo. 2017. Fairness scheduling with dynamic priority for multi workflow on heterogeneous systems. In *Proceedings of the 2nd IEEE International Conference on Cloud Computing and Big Data Analysis*. 404–409. DOI: http://dx.doi.org/10.1109/ICCCBDA.2017.7951947

[54] Guoqi Xie, Liangjiao Liu, Liu Yang, and Renfa Li. 2017. Scheduling trade-off of dynamic multiple parallel workflows on heterogeneous distributed computing systems. *Concurr. Comput.: Pract. Exp.* 29, 2 (2017), e3782–n/a. DOI: http://dx.doi.org/10.1002/cpe.3782

[55] Yinglin Tsai, Hsiaoching Liu, and Kuochan Huang. 2015. Adaptive dual-criteria task group allocation for clustering-based multi-workflow scheduling on parallel computing platform. *J. Supercomput.* 71, 10 (2015), 3811–3831. DOI: http://dx.doi.org/10.1007/s11227-015-1469-x

[56] Bing Lin, Wenzhong Guo, and Xiuyan Lin. 2016. Online optimization scheduling for scientific workflows with deadline constraint on hybrid clouds. *Concurr. Comput.: Pract. Exp.* 28, 11 (2016), 3079–3095. DOI: http://dx.doi.org/10.1002/cpe.3582

[57] Shaghayegh Sharif, Javid Taheri, Albert Y. Zomaya, and Surya Nepal. 2014. Online multiple workflow scheduling under privacy and deadline in hybrid cloud environment. In *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science*. 455–462. DOI: http://dx.doi.org/10.1109/CloudCom.2014.128

[58] Maria A. Rodriguez and Rajkumar Buyya. 2018. Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. *Fut. Gener. Comput. Syst.* 79, 2 (2018), 739–750. DOI: http://dx.doi.org/10.1016/j.future.2017.05.009

[59] Xiaomin Zhu, Ji Wang, Hui Guo, Dakai Zhu, Laurence T. Yang, and Ling Liu. 2016. Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. *IEEE Trans. Parallel Distrib. Syst.* 27, 12 (2016), 3501–3517. DOI: http://dx.doi.org/10.1109/TPDS.2016.2543731

[60] Xiaolong Xu, Wanchun Dou, Xuyun Zhang, and Jinjun Chen. 2016. EnReal: An energy-aware resource allocation method for scientific workflow executions in cloud environment. *IEEE Trans. Cloud Comput.* 4, 2 (2016), 166–179. DOI: http://dx.doi.org/10.1109/TCC.2015.2453966

[61] Huangke Chen, Xiaomin Zhu, Dishan Qiu, Hui Guo, Laurence T. Yang, and Peizhong Lu. 2016. EONS: Minimizing energy consumption for executing real-time workflows in virtualized cloud data centers. In *Proceedings of the 45th International Conference on Parallel Processing Workshops*. 385–392. DOI: http://dx.doi.org/10.1109/ICPPW.2016.60

[62] Guoqi Xie, Gang Zeng, Junqiang Jiang, Chunnian Fan, Renfa Li, and Keqin Li. 2017. Energy management for multiple real-time workflows on cyber–physical cloud systems. *Fut. Gener. Comput. Syst.* (2017). DOI: http://dx.doi.org/10.1016/j.future.2017.05.033

[63] Huangke Chen, Xiaomin Zhu, Dishan Qiu, and Ling Liu. 2016. Uncertainty-aware real-time workflow scheduling in the cloud. In *Proceedings of the 9th IEEE International Conference on Cloud Computing*. 577–584. DOI: http://dx.doi.org/10.1109/CLOUD.2016.0082

[64] Huangke Chen, Jianghan Zhu, Zhenshi Zhang, Manhao Ma, and Xin Shen. 2017. Real-time workflows oriented online scheduling in uncertain cloud environment. *J. Supercomput.* 73, 11 (2017), 4906–4922. DOI: http://dx.doi.org/10.1007/s11227-017-2060-4

[65] Huangke Chen, Xiaomin Zhu, Guipeng Liu, and Witold Pedrycz. 2018. Uncertainty-aware online scheduling for real-time workflows in cloud service environment. *IEEE Trans. Serv. Comput.* (2018), 1–1. DOI: http://dx.doi.org/10.1109/TSC.2018.2866421

[66] Jiagang Liu, Ju Ren, Wei Dai, Deyu Zhang, Pude Zhou, Yaoxue Zhang, Geyong Min, and Noushin Najjari. 2019. Online multi-workflow scheduling under uncertain task execution time in IaaS clouds. *IEEE Trans. Cloud Comput.* (2019), 1–1. DOI: http://dx.doi.org/10.1109/TCC.2019.2906300

[67] Georgios L. Stavrinides, Francisco R. Duro, Helen D. Karatza, Javier G. Blas, and Jesus Carretero. 2017. Different aspects of workflow scheduling in large-scale distributed systems. *Simul. Model. Pract. Theory* 70, Suppl. C (2017), 120–134. DOI: http://dx.doi.org/10.1016/j.simpat.2016.10.009

[68] Naqin Zhou, FuFang Li, Kefu Xu, and Deyu Qi. 2018. Concurrent workflow budget- and deadline-constrained scheduling in heterogeneous distributed environments. *Soft Computing* 22, 23 (2018), 7705–7718. DOI: http://dx.doi.org/10.1007/s00500-018-3229-3

[69] Huangke Chen, Jianghan Zhu, Guohua Wu, and Lisu Huo. 2018. Cost-efficient reactive scheduling for real-time workflows in clouds. *J. Supercomput.* 74, 11 (2018), 6291–6309. DOI: http://dx.doi.org/10.1007/s11227-018-2561-9

[70] Georgios L. Stavrinides and Helen D. Karatza. 2010. Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations. *J. Syst. Softw.* 83, 6 (2010), 1004–1014. DOI: http://dx.doi.org/10.1016/j.jss.2009.12.025

[71] Francisco R. Duro, Javier G. Blas, and Jesus Carretero. 2013. A hierarchical parallel storage system based on distributed memory for large scale systems. In *Proceedings of the 20th European MPI Users' Group Meeting*. 139–140. DOI: http://dx.doi.org/10.1145/2488551.2488598

[72] Xiaoyong Tang, Kenli Li, Guiping Liao, Kui Fang, and Fan Wu. 2011. A stochastic scheduling algorithm for precedence constrained tasks on grid. *Fut. Gener. Comput. Syst.* 27, 8 (2011), 1083–1091. DOI: http://dx.doi.org/10.1016/j.future.2011.04.007

[73] Deepak Poola, Saurab Garg, Rajkumar Buyya, Yun Yang, and Kotagiri Ramamohanarao. 2014. Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In *Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications*. 858–865. DOI: http://dx.doi.org/10.1109/AINA.2014.105

[74] Vahid Ebrahimirad, Maziar Goudarzi, and Aboozar Rajabi. 2015. Energy-aware scheduling for precedence-constrained parallel virtual machines in virtualized data centers. *J. Grid Comput.* 13, 2 (2015), 233–253. DOI: http://dx.doi.org/10.1007/s10723-015-9327-x

[75] Ilia Pietri and Rizos Sakellariou. 2014. Energy-aware workflow scheduling using frequency scaling. In *Proceedings of the 43rd International Conference on Parallel Processing Workshops*. 104–113. DOI: http://dx.doi.org/10.1109/ICPPW.2014.26

[76] Xiao Qin and Hong Jiang. 2006. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. *Parallel Comput.* 32, 5 (2006), 331–356. DOI: http://dx.doi.org/10.1016/j.parco.2006.06.006

[77] Shuo Zhang, Baosheng Wang, Baokang Zhao, and Jing Tao. 2013. An energy-aware task scheduling algorithm for a heterogeneous data center. In *Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. 1471–1477. DOI: http://dx.doi.org/10.1109/TrustCom.2013.178

[78] Juan J. Durillo, Hamid M. Fard, and Radu Prodan. 2012. MOHEFT: A multi-objective list-based method for workflow scheduling. In *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science*. 185–192. DOI: http://dx.doi.org/10.1109/CloudCom.2012.6427573

[79] Guo Zhong Tian, Chuang Bai Xiao, Zhu Sheng Xu, and Xia Xiao. 2012. Hybrid scheduling strategy for multiple DAGs workflow in heterogeneous system. *J. Softw.* 23, 10 (2012), 2720–2734.

[80] Guoqi Xie, Renfa Li, Xiongren Xiao, and Yuekun Chen. 2014. A high-performance DAG task scheduling algorithm for heterogeneous networked embedded systems. In *Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications*. 1011–1016. DOI: http://dx.doi.org/10.1109/AINA.2014.123

[81] Zhuo Tang, Ling Qi, Zhenzhen Cheng, Kenli Li, Samee U. Khan, and Keqin Li. 2016. An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment. *J. Grid Comput.* 14, 1 (2016), 55–74. DOI: http://dx.doi.org/10.1007/s10723-015-9334-y

[82] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D. Carothers, Kerstin K. van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey Vetter. 2018. The future of scientific workflows. *Int. J. High Perf. Comput. Appl.* 32, 1 (2018), 159–175. DOI: http://dx.doi.org/10.1177/1094342017704893

[83] Maria Fazio, Antonio Celesti, Rajiv Ranjan, Chang Liu, Lydia Chen, and Massimo Villari. 2016. Open issues in scheduling microservices in the cloud. *IEEE Cloud Comput.* 3, 5 (2016), 81–88. DOI: http://dx.doi.org/10.1109/MCC.2016.112

[84] Zhanibek Kozhirbayev and Richard O. Sinnott. 2017. A performance comparison of container-based technologies for the cloud. *Fut. Gener. Comput. Syst.* 68, Suppl. C (2017), 175–182. DOI: http://dx.doi.org/10.1016/j.future.2016.08.025

[85] Wolfgang Gerlach, Wei Tang, Andreas Wilke, Dan Olson, and Folker Meyer. 2015. Container orchestration for scientific workflows. In *Proceedings of the IEEE International Conference on Cloud Engineering*. 377–378. DOI: http://dx.doi.org/10.1109/IC2E.2015.87

[86] Rawaa Qasha, Jacek Cala, and Paul Watson. 2016. Dynamic deployment of scientific workflows in the cloud using container virtualization. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science*. 269–276. DOI: http://dx.doi.org/10.1109/CloudCom.2016.0052

[87] Kai Liu, Kento Aida, Shigetoshi Yokoyama, and Yoshinobu Masatani. 2016. Flexible container-based computing platform on cloud for scientific workflows. In *Proceedings of the International Conference on Cloud Computing Research and Innovations*. 56–63. DOI: http://dx.doi.org/10.1109/ICCCRI.2016.17

[88] Eidah J. Alzahrani, Zahir Tari, Young C. Lee, Deafallah Alsadie, and Albert Y. Zomaya. 2017. adCFS: Adaptive completely fair scheduling policy for containerised workflows systems. In *Proceedings of the 16th IEEE International Symposium on Network Computing and Applications*. 1–8. DOI: http://dx.doi.org/10.1109/NCA.2017.8171362

[89] Theo Combe, Antony Martin, and Roberto D. Pietro. 2016. To Docker or not to Docker: A security perspective. *IEEE Cloud Comput.* 3, 5 (2016), 54–62. DOI: http://dx.doi.org/10.1109/MCC.2016.100

[90] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. 2017. Singularity: Scientific containers for mobility of compute. *PLoS ONE* 12, 5 (2017), 1–20. DOI: http://dx.doi.org/10.1371/journal.pone.0177459

[91] Emily Le and David Paz. 2017. Performance analysis of applications using singularity container on SDSC comet. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*. 66:1–66:4. DOI: http://dx.doi.org/10.1145/3093338.3106737

[92] Heru Suhartanto, Agung P. Pasaribu, Muhammad F. Siddiq, Muhammad I. Fadhila, Muhammad H. Hilman, and Arry Yanuar. 2017. A preliminary study on shifting from virtual machine to Docker container for insilico drug discovery in the cloud. *Int. J. Technol.* 8, 4 (2017).

[93] Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya. 2017. ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers. *Softw.: Pract. Exp.* 47, 4 (2017), 505–521. DOI: http://dx.doi.org/10.1002/spe.2422

[94] Maciej Malawski. 2016. Towards serverless execution of scientific workflows - HyperFlow case study. In *Proceedings of the Workshop of Workflows in Support of Large-Scale Sciences*. 25–33.

[95] Qingye Jiang, Young C. Lee, and Albert Y. Zomaya. 2017. Serverless execution of scientific workflows. In *Proceedings of the 15th International Conference Service-Oriented Computing*. 706–721. DOI: http://dx.doi.org/10.1007/978-3-319-69035-3_51

[96] Maciej Malawski, Adam Gajek, Adam Zima, Bartosz Balis, and Kamil Figiela. 2017. Serverless execution of scientific workflows: Experiments with HyperFlow, AWS lambda and Google cloud functions. *Fut. Gener. Comput. Syst.* (2017). DOI: http://dx.doi.org/10.1016/j.future.2017.10.029

[97] Josef Spillner, Cristian Mateos, and David A. Monge. 2018. FaaSter, better, cheaper: The prospect of serverless scientific computing and HPC. In *High Performance Computing*, Esteban Mocskos and Sergio Nesmachnow (Eds.). Springer International Publishing, Cham, 154–168.

[98] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. 2013. Unikernels: Library operating systems for the cloud. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*. 461–472. DOI: http://dx.doi.org/10.1145/2451116.2451167

[99] Dan Williams, Ricardo Koller, Martin Lucina, and Nikhil Prakash. 2018. Unikernels as processes. In *Proceedings of the ACM Symposium on Cloud Computing*. 199–211. DOI: http://dx.doi.org/10.1145/3267809.3267845

[100] Foued Jrad, Jie Tao, and Achim Streit. 2013. A broker-based framework for multi-cloud workflows. In *Proceedings of the International Workshop on Multi-cloud Applications and Federated Clouds*. 61–68. DOI: http://dx.doi.org/10.1145/2462326.2462339

[101] Javier D. Montes, Mengsong Zou, Rahul Singh, Shu Tao, and Manish Parashar. 2014. Data-driven workflows in multi-cloud marketplaces. In *Proceedings of the 7th IEEE International Conference on Cloud Computing*. 168–175. DOI: http://dx.doi.org/10.1109/CLOUD.2014.32

[102] Yushi Omote, Takahiro Shinagawa, and Kazuhiko Kato. 2015. Improving agility and elasticity in bare-metal clouds. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems*. 145–159. DOI: http://dx.doi.org/10.1145/2694344.2694349

[103] Ryan Shea, Feng Wang, Haiyang Wang, and Jiangchuan Liu. 2014. A deep investigation into network performance in virtual machine based cloud environments. In *Proceeding of the IEEE Conference on Computer Communications*. 1285–1293. DOI: http://dx.doi.org/10.1109/INFOCOM.2014.6848061

[104] Farrukh Nadeem, Daniyal Alghazzawi, Abdulfattah Mashat, Khalid Fakeeh, Abdullah Almalaise, and Hani Hagras. 2017. Modeling and predicting execution time of scientific workflows in the grid using radial basis function neural network. *Cluster Comput.* 20, 3 (2017), 2805–2819. DOI: http://dx.doi.org/10.1007/s10586-017-1018-x

[105] Doyen Sahoo, Steven C. H. Hoi, and Bin Li. 2019. Large scale online multiple kernel regression with application to time-series prediction. *ACM Trans. Knowl. Discov. Data* 13, 1 (2019), 9:1–9:33. DOI: http://dx.doi.org/10.1145/3299875

[106] Muhammad H. Hilman, Maria A. Rodríguez, and Rajkumar Buyya. 2018. Task runtime prediction in scientific workflows using an online incremental learning approach. In *Proceedings of the 11th IEEE/ACM International Conference on Utility and Cloud Computing*. 93–102. DOI: http://dx.doi.org/10.1109/UCC.2018.00018

[107] Jan Zenisek, Florian Holzinger, and Michael Affenzeller. 2019. Machine learning based concept drift detection for predictive maintenance. *Comput. Industr. Eng.* 137 (2019), 106031. DOI: http://dx.doi.org/10.1016/j.cie.2019.106031

[108] Taghrid Samak, Dan Gunter, Monte Goode, Ewa Deelman, Gideon Juve, Gaurang Mehta, Fabio Silva, and Karan Vahi. 2011. Online fault and anomaly detection for large-scale scientific workflows. In *Proceedings of the IEEE International Conference on High Performance Computing and Communications*. 373–381. DOI: http://dx.doi.org/10.1109/HPCC.2011.55

[109] Prathamesh Gaikwad, Anirban Mandal, Paul Ruth, Gideon Juve, Dariusz Król, and Ewa Deelman. 2016. Anomaly detection for scientific workflow applications on networked clouds. In *Proceedings of the International Conference on High Performance Computing Simulation*. 645–652. DOI: http://dx.doi.org/10.1109/HPCSim.2016.7568396

[110] Maria A. Rodriguez, Ramamohanarao Kotagiri, and Rajkumar Buyya. 2018. Detecting performance anomalies in scientific workflows using hierarchical temporal memory. *Fut. Gener. Comput. Syst.* 88 (2018), 624–635. DOI: http://dx.doi.org/10.1016/j.future.2018.05.014

[111] Hamid Arabnejad and Jorge G. Barbosa. 2015. Multi-workflow QoS-constrained scheduling for utility computing. In *Proceedings of the 18th IEEE International Conference on Computational Science and Engineering*. 137–144. DOI: http://dx.doi.org/10.1109/CSE.2015.29

[112] Mozhgan Ghasemzadeh, Hamid Arabnejad, and Jorge G. Barbosa. 2017. Deadline-budget constrained scheduling algorithm for scientific workflows in a cloud environment. In *Proceedings of the 20th International Conference on Principles of Distributed Systems*, Vol. 70. 19:1–19:16. DOI: http://dx.doi.org/10.4230/LIPIcs.OPODIS.2016.19

[113] Hamid M. Fard, Radu Prodan, Juan J. Durillo, and Thomas Fahringer. 2012. A multi-objective approach for workflow scheduling in heterogeneous environments. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 300–309. DOI: http://dx.doi.org/10.1109/CCGrid.2012.114

[114] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. 2015. Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. *Fut. Gener. Comput. Syst.* 48, Suppl. C (2015), 1–18. DOI: http://dx.doi.org/10.1016/j.future.2015.01.004

[115] Anton Beloglazov, Rajkumar Buyya, Young C. Lee, and Albert Y. Zomaya. 2011. Chapter 3 - A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, Vol. 82. Elsevier, 47–111. DOI: http://dx.doi.org/h10.1016/B978-0-12-385512-1.00003-7

[116] Adel Nadjaran Toosi, Chenhao Qu, Marcos Dias de Assunção, and Rajkumar Buyya. 2017. Renewable-aware geographical load balancing of web applications for sustainable data centers. *J. Netw. Comput. Appl.* 83, C (2017), 155–168. DOI: http://dx.doi.org/10.1016/j.jnca.2017.01.036

[117] Lingfang Zeng, Bharadwaj Veeravalli, and Xiaorong Li. 2015. SABA: A security-aware and budget-aware workflow scheduling strategy in clouds. *J. Parallel Distrib. Comput.* 75 (2015), 141–151. DOI: http://dx.doi.org/10.1016/j.jpdc.2014.09.002

[118] Feng Zhao, Chao Li, and Chunfeng Liu. 2014. A cloud computing security solution based on fully homomorphic encryption. In *Proceedings of the 16th International Conference on Advanced Communication Technology*. 485–488. DOI: http://dx.doi.org/10.1109/ICACT.2014.6779008

[119] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Fut. Gener. Comput. Syst.* 29, 7 (2013), 1645–1660. DOI: http://dx.doi.org/10.1016/j.future.2013.01.010

[120] Charalampos Doukas and Fabio Antonelli. 2014. A full end-to-end platform as a service for smart city applications. In *Proceedings of the 10th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*. 181–186. DOI: http://dx.doi.org/10.1109/WiMOB.2014.6962168

[121] Matteo Nardelli, Stefan Nastic, Schahram Dustdar, Massimo Villari, and Rajiv Ranjan. 2017. Osmotic flow: Osmotic computing + IoT workflow. *IEEE Cloud Comput.* 4, 2 (2017), 68–75. DOI: http://dx.doi.org/10.1109/MCC.2017.22

[122] Georgios L. Stavrinides and Helen D. Karatza. 2019. A hybrid approach to scheduling real-time IoT workflows in fog and cloud environments. *Multimedia Tools Appl.* 78, 17 (2019), 24639–24655. DOI: http://dx.doi.org/10.1007/s11042-018-7051-9

Received July 2019; revised September 2019; accepted October 2019

Online Appendix to: Multiple Workflows Scheduling in Multi-tenant Distributed Systems: A Taxonomy and Future Directions

MUHAMMAD H. HILMAN, MARIA A. RODRIGUEZ, and RAJKUMAR BUYYA, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia

A EXPERIMENTAL DESIGN OF SURVEYED ALGORITHMS

This document contains the experimental settings of each algorithm that were not described in the main article.

Table 1. Experimental Design of RANK_HYBD Algorithm [1]

Experiment Type	Simulation	
Workload	Synthetic workflows	: Taken from Hönig et al. [2]
	Number of workflows	: 25 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (175–249)	: Number of tasks
	Meshing degree	: Nodes connectivity
	Edge-length	: Average number of nodes between two connected nodes
	CCR	: Ratio of computation and communication time
Performance Metrics	Makespan	: Total execution time
	Turnaround time	: Makespan and waiting time
	Resource utilization	: Percentage of time when computational resources are busy

Table 2. Experimental Design of OWM Algorithm [3]

Experiment Type	Simulation	
Grid Settings	Number of processors	: 120 processors
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 100 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (20–100)	: Number of task
	Shape (0.5–2.0)	: Workflow's degree of parallelism
	OutDegree (1–5)	: Maximum number of immediate descendants of a task
	CCR (0.1–2.0)	: Ratio of computation and communication time
	BRange (0.1–1.0)	: Distribution range of computation cost of tasks on processors
	WDAG (100–1,000)	: Average computation cost of a workflow
Performance Metrics	Makespan	: Total execution time
	Schedule length ratio	: Ratio of makespan and critical path length
	Win	: Percentage of a particular workflow got the shortest makespan

0360-0300/2020/02-ART10 $15.00
https://doi.org/10.1145/3368036

Table 3. Experimental Design of MOWS Algorithm [4]

Experiment Type	Simulation	
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 100 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (20–100)	: Number of task
	Shape (0.5–2.0)	: Workflow's degree of parallelism
	OutDegree (1–5)	: Maximum number of immediate descendants of a task
	CCR (0.1–2.0)	: Ratio of computation and communication time
	BRange (0.1–1.0)	: Distribution range of computation cost of tasks on processors
	WDAG (100–1,000)	: Average computation cost of a workflow
Performance Metrics	Turnaround time	: Total execution and waiting time
	Schedule length ratio	: Ratio of makespan and critical path length
	Ratio of Shortest turnaround time	: Ratio of a particular workflow got the shortest turnaround time

Table 4. Experimental Design of P-HEFT Algorithm [5]

Experiment Type	Simulation	
Grid Settings	Number of processors	: 50 processors
	Maximum processor speed	: 400 Mflops/s
	Network bandwidth	: 100 Mbps
	Network latency	: 50 μs
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 12 workflows
	Arrival intervals	: 40% elapsed time of the last job
Workflow Properties	Width	: Number of task on the largest level
	Regularity	: Uniformity of the number of task in each level
	Density	: Number of edges between two levels
	Jumps	: Edge connection between two consecutive levels
Performance Metrics	Makespan	: Total execution time
	Efficiency	: Ratio of sequential execution and parallel time

Table 5. Experimental Design of MQMW Algorithm [6]

Experiment Type	Simulation	
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 25 workflows
Performance Metrics	Makespan	: Total execution time
	Cost	: Cloud resource monetary cost
	Success rate	: Percentage of successfully executed workflows

Table 6. Experimental Design of MQSS Algorithm [7]

Experiment Type	Simulation	
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 30 workflows
Performance Metrics	Relative weight of resource	: Different computational capacity of resources
	Success rate	: Percentage of successfully executed workflows

Table 7. Experimental Design of EDF_BF Algorithm [8]

Experiment Type	Simulation	
Grid Settings	Number of processors	: 32 processors
	Heterogeneity level (0–2)	: Difference in processors' speed
	Mean execution rate of processors ($\overline{\mu}$ =1)	: Processors' speed
	Mean data transfer rate ($\overline{\nu}$ =1)	: Data transfer's speed
Workload	Synthetic workflows	: Randomly generated based on Stavrinides and Karatza [9]
	Number of workflows	: 100 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (1–64)	: Number of tasks
	Relative deadline (1–2)	: Based on critical path length (CPL)
	CCR (0.1–10)	: Ratio of computation and communication time
Performance Metrics	Job guarantee ratio	: Ratio of successfully executed workflows

Table 8. Experimental Design of EDF_BF_IC Algorithm [10]

Experiment Type	Simulation	
Cloud Settings	Number of VMs	: 64 VMs
	Heterogeneity level (0.5)	: Difference in VMs' computational capacity
	Mean VM execution rate ($\overline{\mu}$ =1)	: VM's computational capacity
	Mean data transfer rate ($\overline{v}$ =1)	: Data transfer's speed
	Price per time unit ($0.01)	: VM leased fee per time unit
Workload	Synthetic workflows	: Randomly generated based on Stavrinides and Karatza [11]
	Number of workflows	: 10^5 workflows
	Arrival intervals (λ = 0.2)	: Poisson distribution
Workflow Properties	Node (1–64)	: Number of tasks
	Relative deadline (1–2)	: Based on critical path length (CPL)
	CCR (0.1–10)	: Ratio of computation and communication time
Performance Metrics	Overal provided Quality of Service	: Guarantee ratio × average result precision
	Average cost per application	: Average makespan × price per time unit

Table 9. Experimental Design of CWSA Algorithm [12]

Experiment Type	Simulation using CloudSim [13]	
Cloud Settings	Number of VM type	: 4 types
	Provisioning delay	: 97 s
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 1–20 workflows
Workflow Properties	Node (30–1,000)	: Number of task
	Workflow applications	: CyberShake [15] and SIPHT [16]
	Deadline (2–4)	: Based on critical path length (CPL)
Performance Metrics	Makespan	: Total execution time
	Tardiness	: Exceeding degree of the expected completion time
	Laxity	: Degree of a task's urgency execution
	Mean scheduling execution time	: Average time taken by the scheduler to execute workflow
	Resource utilization rate	: Percentage of time when computational resources are busy
	Makespan standard deviation	: Standard deviation of the workflow's makespan
	Skewness of makespan	: Symmetry measurement of the makespan distribution
	Cost	: Cloud resource monetary cost

Table 10. Experimental Design of FSDP Algorithm [17]

Experiment Type	Simulation	
Workload	Synthetic workflows	: Randomly generated based on Wang et al. [18]
	Number of workflows	: 25 workflows
Workflow Properties	Width (4–12)	: Number of task on the largest level
	Regularity (0.2–0.8)	: Uniformity of the number of task in each level
	Density (0.2–0.8)	: Number of edges between two levels
	Jumps (1–4)	: Edge connection between two consecutive levels
	CCR (0.1–10)	: Ratio of computation and communication time
Performance Metrics	Makespan	: Total execution time
	Win	: Percentage of a particular workflow got the shortest makespan

Table 11. Experimental Design of F_DMHSV Algorithm [19]

Experiment Type	Simulation	
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 50 workflows
	Arrival intervals	: 0–200 time units
Workflow Properties	Node (10–50)	: Number of task
	Shape (0.5–2.0)	: Workflow's degree of parallelism
	OutDegree (1–5)	: Maximum number of immediate descendants of a task
	CCR (0.1–2.0)	: Ratio of computation and communication time
Performance Metrics	Schedule length ratio	: Ratio of makespan and critical path length
	Unfairness	: Difference between a workflow's slowdown and average slowdowns
	Deadline missed ratio	: Ratio of workflows missing the deadline

Table 12. Experimental Design of FDWS Algorithm [20]

Experiment Type	Simulation using SimGrid [21]	
Grid Settings	Number of processors	: 280 processors
	Maximum processor speed	: 13–30 Gflops/s
	Platforms derivation	: Based on Grid5000 deployed in France [22]
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 50 workflows
	Arrival intervals	: 0–90% elapsed time of the last job
Workflow Properties	Width (20–50)	: Number of task on the largest level
	Regularity (0.2–0.8)	: Uniformity of the number of task in each level
	Density (0.2–0.8)	: Number of edges between two levels
	Jumps (1–3)	: Edge connection between two consecutive levels
Performance Metrics	Makespan	: Total execution time
	Turnaround time	: Total execution and waiting time
	Turnaround time ratio	: Ratio of turnaround time and the minimum makespan
	Normalized turnaround time	: Ratio of minimum and actual turnaround time
	Win	: Percentage of a particular workflow got the shortest makespan

Table 13. Experimental Design of Adaptive Dual-criteria Algorithm [23]

Experiment Type	Simulation	
Grid Settings	Number of processors	: 30 processors
Workload	Synthetic workflows	: Randomly generated based on Bharathi et al. [24]
	Number of workflows	: 100 workflows
	Arrival intervals	: 1–1000 s
Workflow Properties	Node (20–30)	: Number of task
	Workflow applications	: LIGO [25]
	CCR (0.1–10)	: Ratio of computation and communication time
Performance Metrics	Makespan	: Total execution time
	Scheduling overhead	: Delay in the scheduling process

Table 14. Experimental Design of MLF_ID Algorithm [26]

Experiment Type	Simulation	
Cloud Settings	Number of Public Cloud VM type	: 6 types
	Number of Private Cloud Instances	: 512 CPUs
Workload	Synthetic workflows	: Randomly generated based on Bharathi et al. [24]
	Number of workflows	: 1000 workflows
Workflow Properties	Node (997–1,000)	: Number of task
	Deadline (0.2–5 hours)	: Based on selected range of instance types for executing tasks
	Input dataset size	: 0–300 GB
Performance Metrics	Deadline met	: Number of workflows met their deadline
	Number of Application	: Number of workflows executed in private clouds
	Cost	: Public cloud resource monetary cost

Table 15. Experimental Design of OPHC-TR Algorithm [27]

Experiment Type	Real experiments	
Workload	Medical research application	: Based on the study by Watson [28]
	Number of workflows	: 1,000 workflows
	Arrival intervals (200–700)	: Poisson distribution
Workflow Properties	Node (9–9,000)	: Number of task
	Deadline (0–1)	: Based on workflow execution time in a dedicated fast resources
	Private instance limitation	: Private cloud resource restriction for a workflow
Performance Metrics	Cost	: Cloud resource monetary cost

Table 16. Experimental Design of Dyna Algorithm [29]

Experiment Type	Simulation using CloudSim [13]	
Cloud Settings	Number of VM type	: 4 types
	Provisioning delay (on-demand instances)	: 120 s
	Provisioning delay (spot instances)	: 420 s
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 100 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (997–1,000)	: Number of task
	Workflow applications	: LIGO [25], Montage [30], and Epigenomics [14]
Performance Metrics	Deadline met	: Number of workflows met their deadline
	Cost	: Cloud resource monetary cost
	Cloud instances type	: Breakdown of cloud instances type during execution

Table 17. Experimental Design of EPSM Algorithm [31]

Experiment Type	Simulation using CloudSim [13]	
Cloud Settings	Number of VM type	: 4 types
	VM Provisioning delay	: 0–250 s
	Container Provisioning delay	: 0–100 s
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 1,000–4,000 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (30–1,000)	: Number of task
	Deadline	: Randomly generated based on simulated execution time
Performance Metrics	Deadline met	: Number of workflows met their deadline
	Cost	: Cloud resource monetary cost
	Avg. VM utilization	: Percentage of time when a VM is busy
	Makespan/Deadline Ratio	: Ratio of actual makespan and assigned deadline
	Cloud instances type	: Breakdown of cloud instances type during execution

Table 18. Experimental Design of DGR Algorithm [32]

Experiment Type	Simulation	
Grid Settings	Number of processors	: 1,000 processors
	Computing capacity	: Normalized value (1–10)
	Computing speed	: Normalized value (1–8)
	Network bandwidth	: Normalized value (1–8)
Workload	Synthetic workflows	: Randomly generated
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (20–100)	: Number of task
	Shape (0.5–2.0)	: Workflow's degree of parallelism
	CCR (0.1–10)	: Ratio of computation and communication time
	Tasks' length (10–800)	: Tasks' execution time based on a certain time units distribution
Performance Metrics	Makespan	: Total execution time
	Makespan speedup	: Makespan improvement percentage against HEFT
	Acceptance rate	: Percentage of successfully executed workflows
	Resource utilization	: Percentage of time when computational resources are busy

Table 19. Experimental Design of FASTER Algorithm [33]

Experiment Type	Simulation using CloudSim [13]	
Cloud Settings	Number of VM type	: 4 types
	VM Provisioning delay	: 105 s
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 400 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (50–500)	: Number of task
	Deadline	: Uniformly distributed based on minimal execution time
Performance Metrics	Guarantee ratio	: Percentage of successfully executed workflows
	Host active time	: Total active time of all hosts
	Ratio of task time over hosts time	: Ratio of tasks' execution time over hosts active time

Table 20. Experimental Design of EnReal Algorithm [34]

Experiment Type	Simulation	
Datacenter Settings	Number of Server type	: 4 types
	Energy consumption rate	: 86–342 W
Workload	Synthetic workflows	: Randomly generated based on Liu et al. [35]
	Number of workflows	: 50–300 workflows
Workflow Properties	Node (5–25)	: Number of task
	Resource requirement (1–15)	: Number of required VMs for each task
	Task length (0.1–5.0)	: Task execution time in hours
Performance Metrics	Resource utilization	: Percentage of time when computational resources are busy
	Energy Consumption	: Datacenter energy consumption

Table 21. Experimental Design of EONS Algorithm [36]

Experiment Type	Simulation using CloudSim [13]	
Datacenter Settings	Number of Server type	: 10 types
	Number of VM types	: 10 types
	CPU resource requirements	: 200–2,000 MHz
	Inter-VM bandwidth	: 1 Gbps
	VM provisioning delay	: 90 s
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (30–100)	: Number of task
	CCR (0.5–5.5)	: Ratio of computation and communication time
Performance Metrics	Resource utilization	: Percentage of time when computational resources are busy
	Energy Consumption	: Datacenter energy consumption

Table 22. Experimental Design of DPMMW & GESMW Algorithm [37]

Experiment Type	Simulation	
Datacenter Settings	Number of processors	: 64 processors
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 10–50 workflows
Workflow Properties	Node (40–55)	: Number of task
	Deadline (0–2)	: Increment of the workflow lower bound execution
	Workflow applications	: Gaussian elimination [38], fast fourier transform [38],
		: linear algebra [39], diamond graph [39], and complete binary tree [39]
Performance Metrics	Deadline missed ratio	: Ratio of workflows missing the deadline
	Energy Consumption	: Datacenter energy consumption

Table 23. Experimental Design of PRS Algorithm [40]

Experiment Type	Simulation using CloudSim [13]	
Cloud Settings	Number of VM type	: 6 types
	Network bandwidth	: 1 Gbps
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 1,000 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (30–100)	: Number of task
	Deadline	: Generated based on the fastest execution time
Performance Metrics	Cost	: Cloud resource monetary cost
	Resource utilization	: Percentage of time when computational resources are busy
	Deviation	: Cost of time deviation between predicted and actual finish time

Table 24. Experimental Design of EDPRS Algorithm [41]

Experiment Type	Simulation using CloudSim [13]	
Cloud Settings	Number of VM type	: 6 types
	VM provisioning delay	: 120 s
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 1000 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (30–100)	: Number of task
	Deadline	: Generated based on the fastest execution time
Performance Metrics	Cost	: Cloud resource monetary cost
	Resource utilization	: Percentage of time when computational resources are busy
	Deviation	: Cost of time deviation between predicted and actual finish time

Table 25. Experimental Design of ROSA Algorithm [42]

Experiment Type	Simulation	
Cloud Settings	Number of VM type	: 4 types
	Network bandwidth	: 1 Gbps
	VM provisioning delay	: 30 s
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 1000 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (25–100)	: Number of task
	Deadline	: Generated based on the fastest execution time
Performance Metrics	Cost	: Cloud resource monetary cost
	Makespan	: Total execution time

Table 26. Experimental Design of NOSF Algorithm [43]

Experiment Type	Simulation	
Cloud Settings	Number of VM type	: 7 types
	Network bandwidth	: 100 Mbps
	VM provisioning delay	: 97 s
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 1,000 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (30–1,000)	: Number of task
	Deadline	: Generated based on the fastest execution time
Performance Metrics	Resource utilization	: Percentage of time when computational resources are busy
	Deadline violation	: Percentage of exceeding workflow deadlines

Table 27. Experimental Design of EDF_BF *In-Mem* Algorithm [44]

Experiment Type	Simulation	
Grid Settings	Number of processors	: 64 processors
	Heterogeneity level (0–2)	: Difference in processors' speed
	Mean execution rate of processors ($\overline{\mu}$ =1)	: Processors' speed
	Mean data transfer rate ($\overline{\nu}$ =1)	: Data transfer's speed
	Mean file system throughput rate ($\overline{\rho}$ =1)	: File system access' speed
Workload	Synthetic workflows	: Randomly generated based on Stavrinides and Karatza [9]
	Number of workflows	: 10^6 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (1–64)	: Number of tasks
	Relative deadline (1–2)	: Based on critical path length (CPL)
	CCR (0.1–10)	: Ratio of computation and communication time
	IOCR (0.25–1)	: Ratio of I/O and communication time
Performance Metrics	Application completion ratio	: Ratio of successfully completed workflows
	Application guarantee ratio	: Ratio of completed workflows within the deadline
	Tardiness	: Exceeding degree of the expected completion time

Table 28. Experimental Design of MW-HBDCS Algorithm [45]

Experiment Type	Simulation using SimGrid [21]	
Grid Settings	Number of processors	: 20–64 processors
	Platforms derivation	: Based on Grid5000 deployed in France [22]
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 50 workflows
	Arrival intervals	: 10–50% elapsed time of the last job
Workflow Properties	Node (10–100)	: Number of task on the largest level
	Regularity (0.2–0.6)	: Uniformity of the number of task in each level
	Density (0.2–0.6)	: Number of edges between two levels
	Jumps (1–4)	: Edge connection between two consecutive levels
	CCR (0.1–5)	: Ratio of computation and communication time
	BRange (0.1–2)	: Distribution range of computation cost of tasks on processors
	Deadline (0.5–0.9)	: Based on min. and max. execution time
	Budget (0.4–0.8)	: Based on min. and max. execution cost
Performance Metrics	Planning successful rate	: Ratio of successfully planed and executed workflows

Table 29. Experimental Design of MW-DBS Algorithm [46]

Experiment Type	Simulation using SimGrid [21]	
Grid Settings	Number of processors	: 20–64 processors
	Platforms derivation	: Based on Grid5000 deployed in France [22]
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 50 workflows
	Arrival intervals	: 10–50% elapsed time of the last job
Workflow Properties	Node (30–100)	: Number of task on the largest level
	Regularity (0.2–0.5)	: Uniformity of the number of task in each level
	Density (0.2–0.5)	: Number of edges between two levels
	Jumps (1–4)	: Edge connection between two consecutive levels
	Deadline (1–2)	: Based on min. and max. execution time
	Budget (0–1)	: Based on min. and max. execution cost
Performance Metrics	Planning successful rate	: Ratio of successfully planed and executed workflows

Table 30. Experimental Design of MQ-PAS Algorithm [47]

Experiment Type	Simulation using SimGrid [21]	
Grid Settings	Number of processors	: 20–64 processors
	Platforms derivation	: Based on Grid5000 deployed in France [22]
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 50 workflows
	Arrival intervals	: 10–50% elapsed time of the last job
Workflow Properties	Node (40–120)	: Number of task on the largest level
	Regularity (0.2–0.8)	: Uniformity of the number of task in each level
	Density (0.2–0.8)	: Number of edges between two levels
	Jumps (1–4)	: Edge connection between two consecutive levels
	Deadline (1–2)	: Based on min. and max. execution time
	Budget (0–1)	: Based on min. and max. execution cost
Performance Metrics	Planning successful rate	: Ratio of successfully planed and executed workflows

Table 31. Experimental Design of CERSA Algorithm [48]

Experiment Type	Simulation	
Cloud Settings	Number of VM type	: 4 types
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [14]
	Number of workflows	: 1,000 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (30–100)	: Number of task
	Deadline	: Generated based on the fastest execution time
Performance Metrics	Cost	: Cloud resource monetary cost
	Resource utilization	: Percentage of time when computational resources are busy

REFERENCES

[1] Zhifeng Yu and Weisong Shi. 2008. A planner-guided scheduling strategy for multiple workflow applications. In *Proceedings of the International Conference on Parallel Processing*. 1–8. DOI: http://dx.doi.org/10.1109/ICPP-W.2008.10

[2] Udo Hönig and Wolfram Schiffmann. 2004. A comprehensive test bench for the evaluation of scheduling heuristics. In *Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems*. 437–442.

[3] Chihchiang Hsu, Kuochan Huang, and Fengjian Wang. 2011. Online scheduling of workflow applications in grid environments. *Fut. Gener. Comput. Syst.* 27, 6 (2011), 860–870. DOI: http://dx.doi.org/10.1016/j.future.2010.10.015

[4] Yirong Wang, Kuochan Huang, and Fengjian Wang. 2016. Scheduling online mixed-parallel workflows of rigid tasks in heterogeneous multi-cluster environments. *Fut. Gener. Comput. Syst.* 60, Suppl. C (2016), 35–47. DOI: http://dx.doi.org/10.1016/j.future.2016.01.013

[5] Jorge G. Barbosa and Belmiro Moreira. 2011. Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters. *Parallel Comput.* 37, 8 (2011), 428–438. DOI: http://dx.doi.org/10.1016/j.parco.2010.12.004

[6] Meng Xu, Lizhen Cui, Haiyang Wang, and Yanbing Bi. 2009. A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications*. 629–634. DOI: http://dx.doi.org/10.1109/ISPA.2009.95

[7] Cui Lizhen, Xu Meng, and Yanbing Bi. 2009. A scheduling strategy for multiple QoS constrained grid workflows. In *Proceedings of the Joint Conferences on Pervasive Computing*. 561–566. DOI: http://dx.doi.org/10.1109/JCPC.2009.5420119

[8] Georgios L. Stavrinides and Helen D. Karatza. 2011. Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. *Simul. Model. Pract. Theory* 19, 1 (2011), 540–552. DOI: http://dx.doi.org/10.1016/j.simpat.2010.08.010

[9] Georgios L. Stavrinides and Helen D. Karatza. 2010. Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations. *J. Syst. Softw.* 83, 6 (2010), 1004–1014. DOI: http://dx.doi.org/10.1016/j.jss.2009.12.025

[10] Georgios L. Stavrinides and Helen D. Karatza. 2015. A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds. In *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud*. 231–239. DOI: http://dx.doi.org/10.1109/FiCloud.2015.93

[11] Georgios L. Stavrinides and Helen D. Karatza. 2014. The impact of resource heterogeneity on the timeliness of hard real-time complex jobs. In *Proceedings of the 7th International Conference on Pervasive Technologies Related to Assistive Environments*. ACM, 65.

[12] Bhaskar P. Rimal and Martin Maier. 2017. Workflow scheduling in multi-tenant cloud computing environments. *IEEE Trans. Parallel Distrib. Syst.* 28, 1 (2017), 290–304. DOI: http://dx.doi.org/10.1109/TPDS.2016.2556668

[13] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Pract. Exp.* 41, 1 (2011), 23–50. DOI: http://dx.doi.org/10.1002/spe.995

[14] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. 2013. Characterizing and profiling scientific workflows. *Fut. Gener. Comput. Syst.* 29, 3 (2013), 682–692. DOI: http://dx.doi.org/10.1016/j.future.2012.08.015

[15] Philip Maechling, Ewa Deelman, Li Zhao, Robert Graves, Gaurang Mehta, Nitin Gupta, John Mehringer, Carl Kesselman, Scott Callaghan, David Okaya, Hunter Francoeur, Vipin Gupta, Yifeng Cui, Karan Vahi, Thomas Jordan, and Edward Field. 2007. *SCEC CyberShake Workflows—Automating Probabilistic Seismic Hazard Analysis Calculations.* Springer London, 143–163. DOI: http://dx.doi.org/10.1007/978-1-84628-757-2_10

[16] Jonathan Livny, Hidayat Teonadi, Miron Livny, and Matthew K. Waldor. 2008. High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs. *PLoS ONE* 3, 9 (09 2008), 1–12. DOI: http://dx.doi.org/10.1371/journal.pone.0003197

[17] Yuxin Wang, Shijie Cao, Guan Wang, Zhen Feng, Chi Zhang, and He Guo. 2017. Fairness scheduling with dynamic priority for multi workflow on heterogeneous systems. In *Proceedings of the 2nd IEEE International Conference on Cloud Computing and Big Data Analysis.* 404–409. DOI: http://dx.doi.org/10.1109/ICCCBDA.2017.7951947

[18] Yuxin Liu, Hui Guo, He Wang, and Guan Wang. 2016. HSIP: A novel task scheduling algorithm for heterogeneous computing. *Scientific Programming* (2016). DOI: http://dx.doi.org/10.1155/2016/3676149

[19] Guoqi Xie, Liangjiao Liu, Liu Yang, and Renfa Li. 2017. Scheduling trade-off of dynamic multiple parallel workflows on heterogeneous distributed computing systems. *Concurr. Comput.: Pract. Exp.* 29, 2 (2017), e3782–n/a. DOI: http://dx.doi.org/10.1002/cpe.3782

[20] Hamid Arabnejad, Jorge G. Barbosa, and Frédéric Suter. 2014. Fair resource sharing for dynamic scheduling of workflows on heterogeneous systems. In *High-Performance Computing on Complex Environments.*

[21] M. Quinson. 2009. SimGrid: A generic framework for large-scale distributed experiments. In *Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing.* 95–96. DOI: http://dx.doi.org/10.1109/P2P.2009.5284500

[22] Franck Cappello, Frédéric Desprez, Michel Dayde, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Nouredine Melab, Raymond Namyst, Pascale Primet, Olivier Richard, Eddy Caron, Julien Leduc, and Guillaume Mornet. 2005. Grid'5000: A large-scale, reconfigurable, controlable and monitorable grid platform. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing.*

[23] Yinglin Tsai, Hsiaoching Liu, and Kuochan Huang. 2015. Adaptive dual-criteria task group allocation for clustering-based multi-workflow scheduling on parallel computing platform. *J. Supercomput.* 71, 10 (2015), 3811–3831. DOI: http://dx.doi.org/10.1007/s11227-015-1469-x

[24] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei Hu Su, and Karan Vahi. 2008. Characterization of scientific workflows. In *Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science.* 1–10. DOI: http://dx.doi.org/10.1109/WORKS.2008.4723958

[25] Kathy Svitil. 2016. Gravitational waves detected 100 years after Einstein's prediction. (2016). Retrieved from http://www.caltech.edu/news/gravitational-waves-detected-100-years-after-einstein-s-prediction-49777.

[26] Bing Lin, Wenzhong Guo, and Xiuyan Lin. 2016. Online optimization scheduling for scientific workflows with deadline constraint on hybrid clouds. *Concurr. Comput.: Pract. Exp.* 28, 11 (2016), 3079–3095. DOI: http://dx.doi.org/10.1002/cpe.3582

[27] Shaghayegh Sharif, Javid Taheri, Albert Y. Zomaya, and Surya Nepal. 2014. Online multiple workflow scheduling under privacy and deadline in hybrid cloud environment. In *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science.* 455–462. DOI: http://dx.doi.org/10.1109/CloudCom.2014.128

[28] Paul Watson. 2012. A multi-level security model for partitioning workflows over federated clouds. *J. Cloud Comput.: Adv. Syst. Appl.* 1 (2012), 1–15. DOI: http://dx.doi.org/10.1186/2192-113X-1-15

[29] Amelia C. Zhou, Bingsheng He, and Cheng Liu. 2016. Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds. *IEEE Trans. Cloud Comput.* 4, 1 (2016), 34–48. DOI: http://dx.doi.org/10.1109/TCC.2015.2404807

[30] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. 2008. The cost of doing science on the cloud: The montage example. In *Proceedings of the ACM/IEEE Conference on Supercomputing.* 1–12. DOI: http://dx.doi.org/10.1109/SC.2008.5217932

[31] Maria A. Rodriguez and Rajkumar Buyya. 2018. Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. *Fut. Gener. Comput. Syst.* 79, 2 (2018), 739–750. DOI: http://dx.doi.org/10.1016/j.future.2017.05.009

[32] Wei Chen, Young C. Lee, Alan Fekete, and Albert Y. Zomaya. 2015. Adaptive multiple-workflow scheduling with task rearrangement. *J. Supercomput.* 71, 4 (2015), 1297–1317. DOI: http://dx.doi.org/10.1007/s11227-014-1361-0

[33] Xiaomin Zhu, Ji Wang, Hui Guo, Dakai Zhu, Laurence T. Yang, and Ling Liu. 2016. Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. *IEEE Trans. Parallel Distrib. Syst.* 27, 12 (2016), 3501–3517. DOI: http://dx.doi.org/10.1109/TPDS.2016.2543731

[34] Xiaolong Xu, Wanchun Dou, Xuyun Zhang, and Jinjun Chen. 2016. EnReal: An energy-aware resource allocation method for scientific workflow executions in cloud environment. *IEEE Trans. Cloud Comput.* 4, 2 (2016), 166–179. DOI: http://dx.doi.org/10.1109/TCC.2015.2453966

[35] X. Liu, Y. Yang, Y. Jiang, and J. Chen. 2011. Preventing temporal violations in scientific workflows: Where and how. *IEEE Trans. Softw. Eng.* 37, 6 (2011), 805–825. DOI: http://dx.doi.org/10.1109/TSE.2010.99

[36] Huangke Chen, Xiaomin Zhu, Dishan Qiu, Hui Guo, Laurence T. Yang, and Peizhong Lu. 2016. EONS: Minimizing energy consumption for executing real-time workflows in virtualized cloud data centers. In *Proceedings of the 45th International Conference on Parallel Processing Workshops*. 385–392. DOI: http://dx.doi.org/10.1109/ICPPW.2016.60

[37] Guoqi Xie, Gang Zeng, Junqiang Jiang, Chunnian Fan, Renfa Li, and Keqin Li. 2017. Energy management for multiple real-time workflows on cyber–physical cloud systems. *Fut. Gener. Comput. Syst.* (2017). DOI: http://dx.doi.org/10.1016/j.future.2017.05.033

[38] Haluk Topcuoglu, Salim Hariri, and Min You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* 13, 3 (2002), 260–274. DOI: http://dx.doi.org/10.1109/71.993206

[39] Keqin Li. 2012. Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers. *IEEE Trans. Comput.* 61, 12 (2012), 1668–1681. DOI: http://dx.doi.org/10.1109/TC.2012.120

[40] Huangke Chen, Xiaomin Zhu, Dishan Qiu, and Ling Liu. 2016. Uncertainty-aware real-time workflow scheduling in the cloud. In *Proceedings of the 9th IEEE International Conference on Cloud Computing*. 577–584. DOI: http://dx.doi.org/10.1109/CLOUD.2016.0082

[41] Huangke Chen, Jianghan Zhu, Zhenshi Zhang, Manhao Ma, and Xin Shen. 2017. Real-time workflows oriented online scheduling in uncertain cloud environment. *J. Supercomput.* 73, 11 (2017), 4906–4922. DOI: http://dx.doi.org/10.1007/s11227-017-2060-4

[42] Huangke Chen, Xiaomin Zhu, Guipeng Liu, and Witold Pedrycz. 2018. Uncertainty-aware online scheduling for real-time workflows in cloud service environment. *IEEE Trans. Serv. Comput.* (2018), 1–1. Early Access. DOI: http://dx.doi.org/10.1109/TSC.2018.2866421

[43] Jiagang Liu, Ju Ren, Wei Dai, Deyu Zhang, Pude Zhou, Yaoxue Zhang, Geyong Min, and Noushin Najjari. 2019. Online multi-workflow scheduling under uncertain task execution time in IaaS clouds. *IEEE Trans. Cloud Comput.* (2019), 1–1. Early Access. DOI: http://dx.doi.org/10.1109/TCC.2019.2906300

[44] Georgios L. Stavrinides, Francisco R. Duro, Helen D. Karatza, Javier G. Blas, and Jesus Carretero. 2017. Different aspects of workflow scheduling in large-scale distributed systems. *Simul. Model. Pract. Theory* 70, Suppl. C (2017), 120–134. DOI: http://dx.doi.org/10.1016/j.simpat.2016.10.009

[45] Naqin Zhou, FuFang Li, Kefu Xu, and Deyu Qi. 2018. Concurrent workflow budget- and deadline-constrained scheduling in heterogeneous distributed environments. *Soft Computing* 22, 23 (2018), 7705–7718. DOI: http://dx.doi.org/10.1007/s00500-018-3229-3

[46] Hamid Arabnejad and Jorge G. Barbosa. 2017. Maximizing the completion rate of concurrent scientific applications under time and budget constraints. *J. Comput. Sci.* 23, Suppl. C (2017), 120–129. DOI: http://dx.doi.org/10.1016/j.jocs.2016.10.013

[47] Hamid Arabnejad and Jorge G. Barbosa. 2017. Multi-QoS constrained and profit-aware scheduling approach for concurrent workflows on heterogeneous systems. *Fut. Gener. Comput. Syst.* 68, Suppl. C (2017), 211–221. DOI: http://dx.doi.org/10.1016/j.future.2016.10.003

[48] Huangke Chen, Jianghan Zhu, Guohua Wu, and Lisu Huo. 2018. Cost-efficient reactive scheduling for real-time workflows in clouds. *J. Supercomput.* 74, 11 (2018), 6291–6309. DOI: http://dx.doi.org/10.1007/s11227-018-2561-9