IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 13, 2020, accepted October 17, 2020, date of publication October 20, 2020, date of current version October 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3032545

Adaptive Resource Allocation and Consolidation for Scientific Workflow Scheduling in Multi-Cloud Environments

ZHEYI CHEN[1], KAI LIN[2], BING LIN[2], XING CHEN[3], XIANGHAN ZHENG[3], AND CHUNMING RONG[4], (Senior Member, IEEE)

[1]College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter EX4 4QF, U.K.
[2]College of Physics and Energy, Fujian Normal University, Fuzhou 350117, China
[3]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China
[4]Department of Electronic Engineering and Computer Science, University of Stavanger, 4036 Stavanger, Norway

Corresponding authors: Bing Lin (WheelLX@163.com) and Xing Chen (chenxing@fzu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 62072108, in part by the Natural Science Foundation of Fujian Province for Distinguished Young Scholar, in part by the Natural Science Foundation of Fujian Province under Grant 2019J01286, in part by the Guiding Project of Fujian Province under Grant 2018H0017, in part by the Young and Middle-aged Teacher Education Foundation of Fujian Province under Grant JT180098, and in part by the China Scholarship Council.

ABSTRACT The emerging multi-cloud environments (MCEs) empower the execution of large-scale scientific workflows (SWs) with sufficient resource provisioning. However, due to complex task dependencies in SWs and various cost-performance of cloud resources, the SW scheduling in MCEs faces huge challenges. To address these challenges, we propose an Online Workflow Scheduling algorithm based on Adaptive resource Allocation and Consolidation (OWS-A2C). In OWS-A2C, the deadline reassignment is first executed for SW tasks based on the execution performance of instance resources, which enhances resource utilization from a local perspective when executing an SW. Next, the execution instances are allocated and consolidated according to the performance requirements of multiple SWs, which improves resource utilization and reduces the total costs of executing multiple SWs from a global perspective. Finally, the SW tasks are dynamically scheduled to the execution instances with the earliest-deadline-first (EDF) discipline and completed before their sub-deadlines. The extensive simulation experiments are conducted to demonstrate the effectiveness of the proposed OWS-A2C on SW scheduling in MCEs, which outperforms three baseline scheduling methods with higher resource utilization and lower execution costs under deadline constraints.

INDEX TERMS Multi-cloud environments, scientific workflows, scheduling optimization, resource allocation, resource consolidation.

I. INTRODUCTION

In recent years, workflow technologies have been widely adopted by many applications in science domains to handle the data analysis and processing for scientific innovation and knowledge discovery [1], [2]. However, due to the ever-increasing data volumes, the performance requirements of many scientific workflows (SWs) on data analysis and processing have significantly increased and thus their executions have been alternatively migrated to the cloud with more powerful computational capacity [3]–[5]. Furthermore, multi-cloud environments (MCEs) have emerged as a new and efficient computing pattern for large-scale SWs [6]–[8], which offer more sufficient resource provisioning and diversified pricing mechanisms. Thus, the MCEs promise a larger optimization space for SW scheduling [9]. However, SWs contain complicated structures with complex task dependencies [10], while the cost-performance of instance resources for executing the same task might be different in MCEs [11]. Therefore, SW scheduling in MCEs can be regarded as an NP-hard problem [12].

In response to this problem, it is necessary to design an effective scheduling strategy to reduce the execution costs of executing SWs while meeting the Quality-of-Service

The associate editor coordinating the review of this manuscript and approving it for publication was Ali Kashif Bashir.

(QoS) requirements. However, most of the current scheduling strategies in cloud environments focus on a single workflow [13]–[16], and they do not fully consider the problem of irregular arrivals in large-scale SW scheduling. Moreover, most of these strategies do not well take some essential constraints into account. For example, the deadline constraint is an important feature in the field of emergency scientific computing [17]. In addition, the existing work mainly depends on the static SW scheduling without the consideration of a real-time manner [18], [19]. Therefore, online scheduling for large-scale SWs with deadline constraints in MCEs has become one of the major challenges in the domain of scientific computing [20].

To address the above challenges, in light of our previous work [21], we propose an Online Workflow Scheduling algorithm based on Adaptive resource Allocation and Consolidation (OWS-A2C). The OWS-A2C aims to improve resource utilization and reduce the execution costs of SWs under deadline constraints in MCEs. The main contributions of this paper are summarized as follows.

- The deadline reassignment is first implemented according to the execution performance of instance resources. From a local perspective, it can improve resource utilization when executing an SW. Next, the execution instances are allocated and consolidated based on the performance requirements of multiple SWs. From a global perspective, it can enhance resource utilization and reduce the total costs of executing multiple SWs. Finally, the SW tasks are dynamically scheduled to the execution instances with the EDF discipline while meeting their deadline constraints.
- Extensive simulation experiments are conducted to demonstrate the effectiveness of the proposed OWS-A2C on SW scheduling in MCEs. The results show that the proposed method can achieve higher resource utilization and lower execution costs than three baseline scheduling methods under deadline constraints.

The rest of this paper is organized as follows. In Section II, we review the related work. Section III formulates the problem of SW scheduling in MCEs. In Section IV, we present the proposed OWS-A2C method in detail. Section V evaluates the proposed method by simulation experiments. In Section VI, we conclude this paper and look for future work.

II. RELATED WORK

As an effective measure for optimizing scientific computing, workflow scheduling has attracted much research attention, while many scholars have contributed to addressing this important problem. In this section, we review the related work on workflow scheduling problem with deadline constraints.

Towards a single workflow, Sakellariou *et al*. [14] proposed a scheduling algorithm for optimizing costs with budget constraints in grid environments, which can adjust the scheduling plan to approach the optimal one by using the most cost-effective allocation strategy. The algorithm has a certain enlightening effect on our proposed work, but it focused on grid environments without considering the resource pricing mechanisms in cloud computing. Abrishami *et al*. [15] extended the workflow scheduling scenarios from the grid to cloud environments and designed two workflow scheduling algorithms, including the IaaS Cloud Partial Critical Paths (IC-PCP) and the IaaS Cloud Partial Critical Paths with Deadline Distribution (IC-PCPD2). This work considered some characteristics of cloud computing, such as on-demand resource allocation and interval-based pricing mechanisms. However, it only focused on the scheduling problem for a single workflow but did not well discuss the scheduling problem for multiple workflows. Based on the Pareto theory, Durillo *et al*. [16] proposed a Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT) algorithm in cloud environments, which optimized workflow execution time and resource utilization. Moreover, Wang *et al*. [22] designed a CLoud scientific wOrkflow SchedUling algoRithm based on attack-defensE game model (CLOSURE), which reduced the attacker's benefits and the time costs of the algorithm. Different from these two work, we consider optimizing resource utilization and workflow execution costs with deadline constraints in MCEs. Arabnejad *et al*. [23] designed a heuristic Budget Deadline Aware Scheduling (BDAS) algorithm targeted for a single workflow with budget and deadline constraints in clouds, which made a balance between budget and deadline constraints and thus improved the success rate of workflow scheduling while optimizing the execution costs. However, this work only considered the scheduling problem for a single workflow. In our previous work [18], we adopted the PCP theory to handle the scheduling problem for a single SW with deadline constraints in MCEs, which optimized the execution costs by compressing data communication. Moreover, we developed an Adaptive Discrete Particle Swarm Optimization with Genetic Algorithm (ADPSOGA) in [24] for scheduling a single SW with deadline constraints in MCEs, which can reduce the execution costs while meeting deadline constraints. However, they only focused on a single workflow, the scheduling problem for multiple SWs was not well considered.

In response to multiple SWs, Malawski *et al*. [19] designed a dynamic scheduling algorithm, which took into account the uncertainty of task execution time and the start-up delay of instances, to guarantee the completion rate of scheduling SWs under budget and deadline constraints. However, this work only relied on one instance type but not multiple ones in MCEs. Lorido-Botran *et al*. [25] and Kang *et al*. [26] improved resource utilization and reduced execution costs based on the principles of instance scheduling scaling and rule scaling, respectively, which adaptively turned on or off instances while meeting the performance requirements of SWs. To some extent, these two works enlighten the instance allocation and consolidation proposed in our work, but they did not consider deadline constraints. Mao and Humphrey [27] proposed an adaptive resource scaling

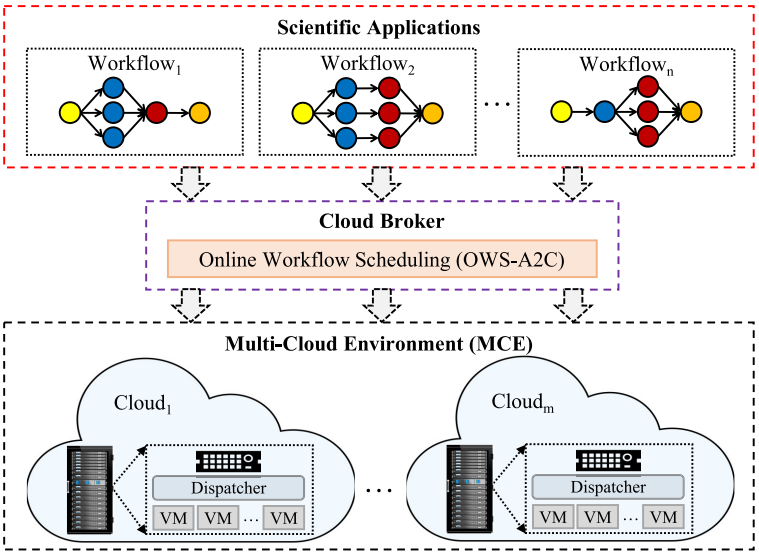


FIGURE 1. The proposed online workflow scheduling model.

method that can reduce the total cost of executing instances. Moreover, Wangsom *et al*. [28] developed a new scheduling framework to optimize network utilization and energy consumption in a cloud data center. However, these two work only targeted at a single cloud environment but did not consider the problems of instance allocation and task scheduling in MCEs. Based on the equal-weight task division in multiple SWs, we designed an online fault-tolerant scheduling strategy in our previous work [21], which can guarantee the high success rate of completing multiple SWs in hybrid cloud environments. By contrast, the objective of this work is to enhance the utilization of instance resources and reduce the total costs of executing multiple SWs in MCEs.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we introduce the proposed online workflow scheduling model in detail. As shown in Figure 1, the proposed model consists of multiple scientific workflows (SWs), a multi-cloud environment (MCE), and a cloud broker who serves as the online workflow scheduler. In the proposed model, multiple SWs with deadline constraints are scheduled by the proposed OWS-A2C, where the SW tasks are distributed to the appropriate execution instances while the idle ones are consolidated adaptively. Thus, not only the utilization of instance resources is improved but also the total execution costs are reduced. Meanwhile, the proposed model ensures that each SW can be completed before their sub-deadlines.

We assume that an SW w_i is described by using a directed acyclic graph (DAG), denoted by $G_i(Vertex_i, Edge_i)$, where $Vertex_i = \{t_{i1}, t_{i2}, \ldots, t_{in}\}$ and $Edge_i = \{e_{i12}, e_{i13}, \ldots, e_{ijk}\}$ are the sets of task nodes and dependencies, respectively. More specifically, $e_{ijk} = (t_{ij}, t_{ik})$ indicates that there is a dependency between the tasks t_{ij} and t_{ik}, where t_{ik} is the child task of t_{ij} and t_{ij} is the parent task of t_{ik}. In this work, we focus on the impact of adaptive resource allocation and consolidation on SW scheduling, and thus the data transmission between different tasks is neglectable while the storage capacity is assumed to be unlimited [29], [30]. During the scheduling process, a child task can only be executed after all its parent tasks have been completed. Moreover, each SW has its deadline constraint, denoted by $D(w_i)$, which represents the expected completion time of an SW.

In an MCE, there are multiple cloud service providers (CSPs), denoted by $P = \{p, q, r, \ldots\}$. For a CSP p, it can provide different types of instances, denoted by $S_p = \{s_{p1}, s_{p2}, \ldots, s_{pm}\}$. We define the execution time of the task t_{ij} on the instance s_{pk} as $T_{exe}(t_{ij}, s_{pk})$, and it is assumed to be known. Meanwhile, the cost-performance might be different when tasks are executed on various types of instances. Moreover, we assume that all instances are equipped with single-core and perform serial processing, and thus only one task can be executed at a time. Besides, a pricing unit time (denoted by λ_p) is specifically set by the CSP p for providing services, and the instance s_{pk} is set a corresponding price in unit time (denoted by c_{pk}).

As the objective of our work is to optimize the computational costs of execution instances, other aspects of costs such as storage, retrieval, analysis, and task conversion are not taken into account [18]. We define the scheduling plan as $S = (Re, Map, T_{total}, C_{total})$, where $Re = \{vm_1, vm_2, \ldots, vm_r\}$ is the set of activated instances, $Map = \{(t_{ij}, vm_k) | t_{ij} \in Vertex_i, vm_k \in Re\}$ is the mapping relationship between tasks and execution instances, T_{total} is the completion time of SWs, and C_{total} is the total execution costs. Moreover, the moments of turning on and off for each execution instance are denoted by $Tls(vm_i)$ and $Tle(vm_i)$, respectively. When the task t_{ij} is completed, its actual end and start moments are recorded, denoted by $AET(t_{ij})$ and $AST(t_{ij})$, respectively.

Therefore, the total execution costs are defined as

$$C_{total} = \sum_{i=1}^{|Re|} c_{s(vm_i)} \cdot \lceil \frac{Tle(vm_i) - Tls(vm_i)}{\lambda_{p(vm_i)}} \rceil, \quad (1)$$

where $s(vm_i)$ is the instance type, $c_s(vm_i)$ is the corresponding unit-time price, $p(vm_i)$ is the CSP who provides the instance, and $\lambda_{p(vm_i)}$ is the pricing unit time of the CSP.

Since many execution instances might be activated for meeting the service requirements of multiple SWs, our optimization objective is to improve the average resource utilization of execution instances. Based on the above definitions, we formulate the online scheduling problem for multiple SWs with deadline constraints in MCEs as

$$\begin{aligned} &\textbf{M}aximize \ \ U_{instance} \\ &\textbf{M}inimize \ \ C_{total} \\ &\textbf{S}ubject \ to \ \ \forall w_i, \max_{t_{ij} \in w_i} \{ATE(t_{ij})\} \leq D(w_i), \end{aligned} \quad (2)$$

where $U_{instance}$ indicates the average resource utilization of execution instances, and our optimization objective is to maximize $U_{instance}$ and minimize C_{total} while meeting deadline constraints.

IV. DESIGN OF THE OWS-A2C

In this section, we present the proposed Online Workflow Scheduling algorithm based on Adaptive resource Allocation and Consolidation (OWS-A2C). The proposed method can be used to schedule multiple SWs in real-time and effectively allocate and consolidate instance resources based on the system status of MCEs. In MCEs, pricing mechanisms rely on unit intervals without considering the actual execution time of instances. Therefore, it is necessary to consolidate instances that are not fully utilized. First of all, we reassign task deadlines according to the execution performance of instances, which can improve resource utilization from a local perspective when executing an SW. Next, the instances are allocated and consolidated based on the performance requirements of multiple SWs, which can enhance resource utilization and reduce the total costs of executing SWs from a global perspective. Finally, the SW tasks are dynamically scheduled to execution instances by following the EDF discipline while ensuring that all tasks can be completed before their sub-deadlines. More details are given in the following sub-sections.

A. DEADLINE REASSIGNMENT

For an SW, it has its own deadline constraint. Before scheduling, the tasks in an SW need to be divided so that they can be scheduled independently (without dependencies). Next, the deadline of the SW is reassigned to the tasks, which forms their sub-deadlines. If all tasks in an SW are completed before their sub-deadlines, the SW would be completed under its deadline constraint.

The key steps of the deadline reassignment are shown in Algorithm 1.

Algorithm 1 Deadline Reassignment

1: **Procedure** *ReassignDll*($G_i(Vertex_i, Edge_i)$, $D(w_i)$)
2: Confirm the valid instance types in an MCE.
3: **for** each t_{ij} in w_i **do**
4: Schedule t_{ij} to vm_{pkr} with the highest cost-performance.
5: **end for**
6: Form a hypothetical scheduling plan *Map*.
7: Call *Pa2Se*($G_i(Vertex_i, Edge_i)$, $D(w_i)$, *Map*).
8: **while** true **do**
9: **if** $makespan(Map) \leq D(w_i)$ **then**
10: **return** *Map*.
11: **else**
12: **for** each t_{ij} in w_i **do**
13: $Map_{ij} = Map - (t_{ij} \rightarrow vm_{pkr}) + (t_{ij} \rightarrow nextFasterVM)$.
14: $rank_{ij} = \frac{makespan(Map) - makespan(Map_{ij})}{cost(Map_{ij}) - cost(Map)}$.
15: **end for**
16: $index = subscript(\max(rank_{ij}))$.
17: $Map = Map_{index}$.
18: **end if**
19: **end while**
20: **End Procedure**

First of all, the valid instance types in an MCE need to be confirmed, and the tasks in an SW are scheduled to the corresponding instances with the highest cost-performance, and thus a hypothetical scheduling plan is formed (Lines 2∼6), denoted by *Map*. This plan only represents the theoretical calculation and serves as a reference for subsequent actual scheduling. Moreover, it should be noted that one instance may need to execute tasks from multiple different SWs simultaneously in the multi-workflow scheduling problem. Next, parallel tasks are converted to serial ones (Line 7). This is because there might be a large amount of remaining execution time when multiple parallel small tasks occupy execution instances alone [31]. If the makespan of *Map* does not exceed the deadline of the corresponding SW, *Map* will be output (Lines 9∼10). Otherwise, *Map* will be updated iteratively (scheduling tasks to execution instances with faster processing speed). Finally, among the recorded feasible plans (meeting deadline constraints), the plan with the highest cost-performance rank (denoted by Map_{index}) will be selected and output (Lines 12∼17). The rank is defined as

$$rank = \frac{makespan(Map') - makespan(Map)}{cost(Map) - cost(Map')}, \quad (3)$$

where the functions *makespan*() and *cost*() are used to calculate the makespan and execution costs of a scheduling plan, respectively.

Algorithm 2 shows the conversion from parallel to serial tasks in detail, which can be used to improve the resource

Algorithm 2 Conversion From Parallel to Serial Tasks

1: **Procedure** $Pa2Se(G_i(Vertex_i, Edge_i), D(w_i), Map)$
2: **for** each t_{ij} and t_{ik} in w_i **do**
3: **if** t_{ij} and t_{ik} have the same parent and child nodes && $T_{exe}(t_{ij}, s_{pk}) \leq \frac{1}{2}\lambda_p$ && $T_{exe}(t_{ik}, s_{qr}) \leq \frac{1}{2}\lambda_q$ **then**
4: Convert t_{ij} and t_{ik} to serial tasks and schedule them to instances with the fastest processing speed.
5: **if** Map can not satisfy deadline constraint **then**
6: Rollback to the previous status Map'.
7: **end if**
8: **end if**
9: **end for**
10: **End Procedure**

utilization of execution instances. First of all, if the parallel tasks have the same parent and child nodes and their execution time does not exceed half of their execution instances' pricing unit time, they will be converted to serial tasks and scheduled to the instances with the fastest processing speed (Lines 3∼4). During this process, the execution time of a scheduling plan might exceed the deadline. When this happens, the rollback operation will be taken and the scheduling status will return to the previous one that satisfies the deadline constraint (Lines 5∼6).

Taking Figure 2 as an example, the SW occupies 5 execution instances, where the tasks t_3, t_4, t_5, and t_6 have the same parent node t_1 and child node t_7, and each of these parallel tasks occupies one execution instance, respectively. By following Algorithm 2, the tasks t_3, t_4, t_5, and t_6 are converted to serial tasks and scheduled to vm_3, where vm_3 is assumed to offer the fastest processing speed. Thus, the costs of two execution instances (i.e. vm_4 and vm_5) can be saved.

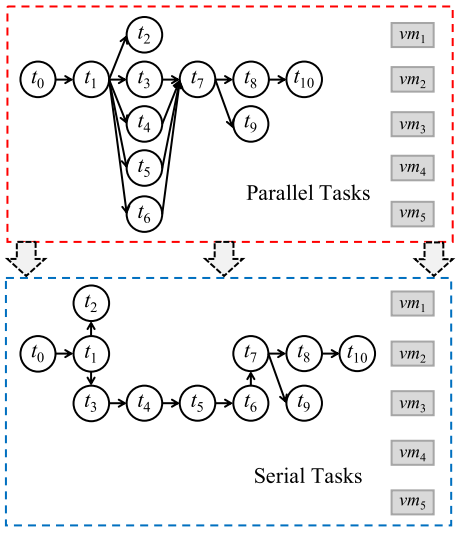


FIGURE 2. An example of the conversion from parallel to serial tasks.

B. INSTANCE ALLOCATION AND CONSOLIDATION

1) INSTANCE ALLOCATION

After the deadline reassignment, each task has a corresponding execution interval, which is defined as

$$Interval_{ij}(T_{start}, T_{end}) = T_{end} - T_{start}, \tag{4}$$

where T_{start} and T_{end} represent the start and end time of the task execution, respectively.

Moreover, we define an execution vector for the SW w_i corresponding to a certain instance type, denoted by $EV(w_i, s_{pk}) = \{ev(t_{i1}, s_{pk}), ev(t_{i2}, s_{pk}), \ldots, ev(t_{in}, s_{pk})\}$, where $ev(t_{ij}, s_{pk})$ indicates the number of instances of the type s_{pk} required to complete the task t_{ij}, which is defined as

$$ev(t_{ij}, s_{pk}) = \frac{T_{exe}(t_{ij}, s_{pk})}{Interval_{ij}(T_{start}, T_{end})}. \tag{5}$$

When $ev(t_{ij}, s_{pk}) > 1$, the execution time exceeds the corresponding execution interval, and thus the task t_{ij} cannot be completed within its sub-deadline by using the current instance type. For example, after deadline reassignment, the execution interval of the task t_{ij} is $Interval_{ij}(5:00, 6:00)$. We assume that the execution time of this task on the instance vm_{pk} is 30 minutes, and thus the corresponding value of $ev(t_{ij}, s_{pk})$ would be $\frac{1}{2}$. Since this value does not exceed 1, the task t_{ij} can be scheduled to the instance vm_{pk} and completed within its execution interval.

Besides, the execution vectors for the SW w_i corresponding to all instance types can be expressed by using the matrix S_{EV} as

$$S_{EV} = \begin{bmatrix} ev(t_{i1}, s_{pk}), ev(t_{i2}, s_{pk}), \ldots, ev(t_{in}, s_{pk}) \\ ev(t_{i1}, s_{qk}), ev(t_{i2}, s_{qk}), \ldots, ev(t_{in}, s_{qk}) \\ ev(t_{i1}, s_{rk}), ev(t_{i2}, s_{rk}), \ldots, ev(t_{in}, s_{rk}) \\ \vdots \end{bmatrix}. \tag{6}$$

By adding the row vectors of S_{EV}, the number of execution instances of different types required to complete the SW w_i before its deadline $D(w_i)$ can be obtained, denoted by $Np = \{N_{pk}, N_{qk}, N_{rk}, \ldots\}$, where N_{pk} is the number of execution instances of the type s_{pk}. During the scheduling process, if the allocated number of execution instances is always greater than or equal to the required ones, all SW tasks would be completed before their sub-deadlines.

2) INSTANCE CONSOLIDATION

An expected scheduling plan is able to schedule tasks to the execution instances with the highest cost-performance, while the instances can achieve high resource utilization. However, due to the different execution time of tasks, the resource utilization of some execution instances may be low in the scheduling process. To address this problem, it is necessary to perform instance consolidation and migrate some tasks to the instances that are not with the highest cost-performance, which can help improve resource utilization and reduce execution costs. As shown in Figure 3, for example, the tasks t_{i1}

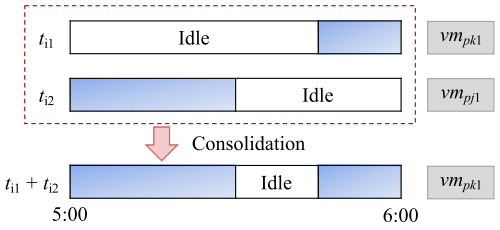


FIGURE 3. An example of the instance consolidation.

and t_{i2} are scheduled to the instances vm_{pk1} and vm_{pj1}, respectively. However, these two tasks only occupy a small part of the pricing unit interval while there are no other tasks to be executed in this interval. Therefore, they are consolidated and scheduled to the instance vm_{pk1}. Although the instance vm_{pk1} may not offer the highest cost-performance to the task t_{i2}, the execution costs of the instance vm_{pj1} in a unit time can be saved by instance consolidation. During the process, each task is guaranteed to be completed before their deadlines.

The key steps of the instance consolidation are shown in Algorithm 3. First of all, the algorithm checks different types of instance resources (denoted by Re) activated in the current MCE and the number of different types of instances (denoted by Re_{pk}) (Line 2). If the required number of execution instances of the type s_{pk} is greater than the currently activated ones, the algorithm will search for other instance types with remaining resources and schedule the uncompleted tasks to these instances (Lines 4∼10), where $t_{top\_in\_s_{pk}}$ represents the task that is closest to its sub-deadline. Finally, the algorithm keeps running until the number of activated instances exceeds the required ones (Lines 11∼12).

Algorithm 3 Instance Consolidation

1: **Procedure** *ConsolidateIns*(Np, Re)
2: Check the activated instance types Re and the number of instances Re_{pk}.
3: **for** each s_{pk} **do**
4: **if** $N_{pk} > Re_{pk}$ **then**
5: **for** each s_{qk} **do**
6: **if** $N_{qk} + ev(t_{top\_in\_s_{pk}}, s_{qk}) \leq Re_{qk}$ **then**
7: $N_{pk} = N_{pk} - ev(t_{top\_in\_s_{pk}}, s_{pk})$.
8: $N_{qk} = N_{qk} + ev(t_{top\_in\_s_{pk}}, s_{qk})$.
9: Schedule $t_{top\_in\_s_{pk}}$ to an instance of s_{qk}.
10: **end if**
11: **if** $N_{pk} \leq Re_{pk}$ **then**
12: **break**.
13: **end if**
14: **end for**
15: **end if**
16: **end for**
17: **End Procedure**

3) ONLINE SCHEDULING FOR MULTIPLE SWs

Through the instance allocation and consolidation, the required number of different types of execution instances can be determined. Furthermore, based on the earliest-deadline-first (EDF) discipline [32], the online scheduling is performed for the tasks whose deadlines have been reassigned. For a certain type of execution instances, all the tasks on the corresponding instances are sorted by their sub-deadlines. When an instance is available, the tasks will be scheduled to this instance by following the EDF discipline.

More specifically, the flow of the proposed OWS-A2C is shown in Figure 4, where the detailed steps are listed as follows.

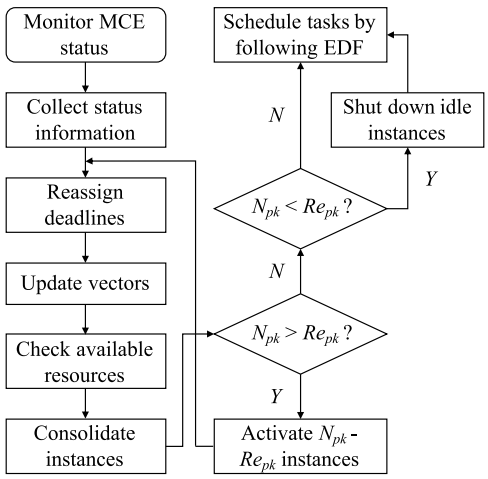


FIGURE 4. The flow of the proposed OWS-A2C.

Step 1: Monitor and collect the status information such as valid instance types, activated execution instances, and corresponding task execution progress in an MCE.

Step 2: According to Algorithm 1, reassign the task deadlines in SWs and calculate their corresponding execution intervals. To reduce resource waste, Algorithm 2 is used to convert parallel tasks to serial ones.

Step 3: Update the execution vectors corresponding to different instance types for each task, and generate the execution matrix S_{EV} for each SW.

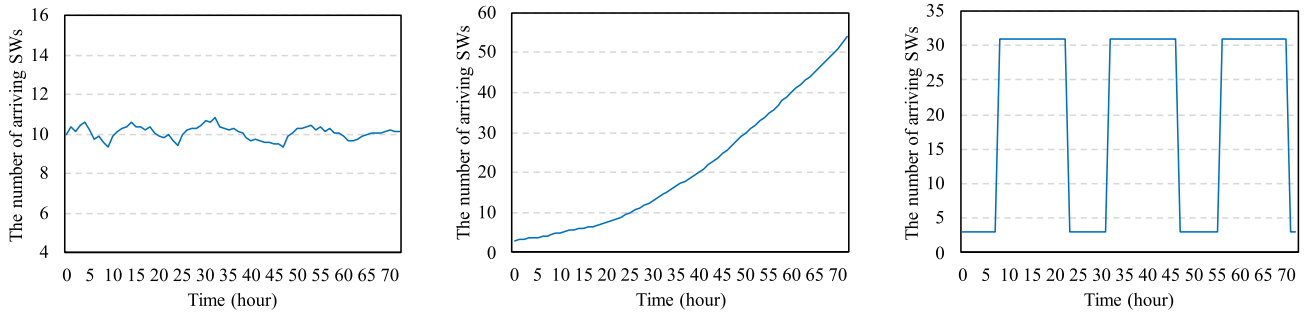
Step 4: Check the execution status and available resources of the activated execution instances, and perform instance consolidation by using Algorithm 3.

Step 5: Determine whether the existing instance resources are sufficient. If the current instance resources are insufficient, activate the required number of execution instances and skip to ***Step 2***. If the current instance resources are sufficient, further determine whether the resource waste happens. If the current instance resources are surplus, shut down idle execution instances. Otherwise, skip to ***Step 6***.

Step 6: Based on the current status of SW tasks and execution instances, schedule the tasks by following the EDF principle.

V. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the proposed OWS-A2C for SW scheduling and make comparisons with three baseline scheduling methods.



(a) Steady-load mode. (b) Growing-load mode. (c) Switching-load mode.

FIGURE 5. **Different load modes for multiple SWs.**

A. SETTINGS AND DATASETS

The simulation environment is established on Windows 7 64-bit with Intel® Core™ i7 CPU @2.30 GHz and RAM 8.00 GB DDR4. Specifically, we implement the multi-cloud simulation environments and the proposed scheduling method for multiple SWs based on Python 3.6, where NumPy is used to provide massive mathematical function libraries for array and matrix operations. Moreover, five types of SWs are used in our simulations including Montage, CyberShake, Epigenomics, LIGO, and SIPHT [33], where each one has its unique structure. For example, there are many parallel tasks in LIGO and SIPHT without obvious task dependency. By contrast, there are many serial tasks in Epigenomics with many task dependencies. For each SW type, a medium-size SW with around 100 tasks is chosen.

Next, three representative load modes are selected for each SW type [34], including the steady-load, growing-load, and switching-load modes. As shown in Figure 5, we simulate the arrival pattern and number of multiple SWs within 72 hours, where the time interval of SWs' arrivals is 5 minutes. In the steady-load mode, the number of arriving SWs is stable with little fluctuations. In the growing-load mode, the number of arriving SWs grows rapidly. In the switching-load mode, the number of arriving SWs shows a strong time correlation.

Moreover, we assume that there are three CSPs (i.e. C1, C2, and C3) in an MCE, where each CSP offers 8 instance types with the corresponding processing speed and execution cost per unit time. Among the types provided by C1(C2/C3), the fastest processing speed of a type is approximately 5(8/10) times the slowest one. Correspondingly, the execution cost increases by the same multiple [18]. Specifically, the execution cost per unit time of an instance type with the slowest processing speed in each CPS is set to 2 USD/hour, and the pricing unit time is set to 1 hour [35]. Furthermore, the task execution time of the instance with the slowest processing speed is the same as that generated by the workflow generator proposed in [36], which facilitates the evaluation of scheduling systems and algorithms for different SW sizes and parameterizes the information collected from the actual executions of SWs to generate synthetic SWs that can resemble those used by real-world scientific applications. In addition, we assume that about one-third of tasks in an SW can be executed on the instances with the corresponding highest cost-performance [23].

Finally, we define five different deadlines for each SW as

$$D_k(w_i) = r_k \cdot \min(w_i), \tag{7}$$

where $k = \{1, 2, \ldots, 5\}$, $\min(w_i)$ is the time required for completing the SW w_i by using the Heterogeneous Earliest-Finish-Time (HEFT) algorithm [37], and the value of r_k is sequentially taken from the set $R = \{1.2, 1.5, 3, 5, 8\}$.

To validate the effectiveness of the proposed OWS-A2C, we first modify three scheduling algorithms proposed in the related work so that they can adapt to the current scenario. Next, we regard them as baseline scheduling methods and conduct comparative experiments. The detailed descriptions are given as follows.

- GAINM. By extending the GAIN algorithm [14], the GAIN with Modification GAINM) algorithm can better adapt to the cost optimization problem of scheduling multiple SWs with deadline constraints in MCEs. Firstly, the GAINM select the instance type with the lowest execution cost to schedule the tasks of a single SW. Next, the scheduling plan is continuously updated according to Equation (3) until the execution time satisfies the deadline constraints.
- PHGS. By referring to our previous work [21], the Partition Hierarchically and Greedy Scheduling (PHGS) algorithm first estimates the earliest start time and sub-deadlines of unscheduled tasks based on the equal-weight task division. Next, the tasks are scheduled to the execution instances that satisfy the sub-deadline constraints and consume the lowest execution cost by the greedy algorithm.
- PHA2CI. The deadline reassignment relies on SW structures and task weights by using the equal-weight task division [21], but it does not consider instance types. Therefore, the Partition Hierarchically based on the Adaptive Allocation and Consolidation for Instances (PHA2CI) algorithm is designed to better evaluate the

impact of deadline reassignment on scheduling results, because it performs instance allocation and consolidation during the scheduling process.

B. EXPERIMENTAL RESULTS

As load modes and the arriving number of multiple SWs might be various, we use an average execution cost (denoted by $Avg_{Nec(W)}$) and average utilization of execution instances (denoted by $Avg_{Uti(VM)}$) as the performance indexes for evaluating different scheduling methods, which are defined as

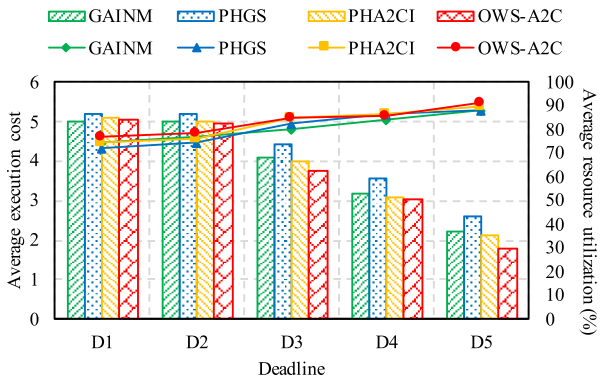
$$Avg_{Nec(W)} = \frac{C_{total}(W)}{N(W) \cdot \min_{w_i \in W}\{CC(w_i)\}}, \quad (8)$$

$$Avg_{Uti(VM)} = \frac{\sum_{i=1}^{M} Uti(vm_i)}{M}, \quad (9)$$

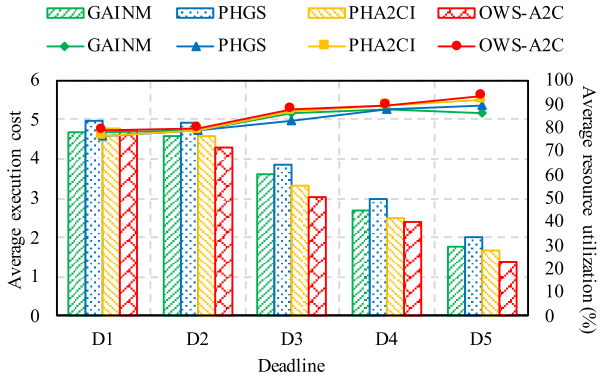
where $C_{total}(W)$ represents the total execution costs and $N(W)$ is the number of arriving multiple SWs. $\min_{w_i \in W}\{CC(w_i)\}$ indicates the lowest cost of scheduling each SW to the corresponding instances by using the greedy algorithm in a single cloud environment, which does not consider deadline constraints. Moreover, M and $Uti(vm_i)$ represent the number of activated execution instances and the resource utilization of each instance during the scheduling process, respectively.

Figures 6, 7, and 8 show the average execution cost and utilization of execution instances for scheduling three types of SWs (i.e. CyberShake, Epigenomics, and LIGO) under three load modes (i.e. steady-load, growing-load, and switching-load modes), respectively. In general, these scheduling methods achieve their corresponding best performance in the growing-load mode, followed by the steady-load and switching-load modes. This is because the number of arriving SWs in the growing-load mode is constantly increasing, and thus there are fewer idle instances, which leads to higher resource utilization and lower average execution cost. By contrast, numerous execution instances need to be activated during the peak period under the switching-load mode, and thus many idle instances might occur after this period, which will result in lower resource utilization and higher average execution cost.

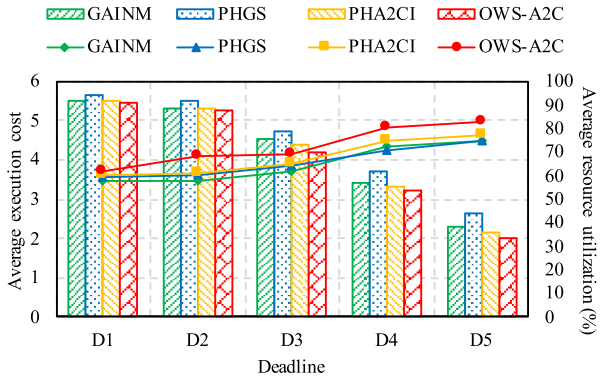
Compared to other scheduling methods, the proposed OWS-A2C shows the best performance, while the PHGS performs worst in most cases. This is because the PHGS performs the deadline reassignment only based on the task loads, and thus it is hard to find the best execution instance to satisfy the reassigned sub-deadlines. Meanwhile, the PHGS uses the greedy algorithm to select scheduling plans, which may save some execution costs when dealing with a single SW. However, when it comes to multiple SWs, this method will cause higher execution costs. By contrast, the PHA2CI performs the instance consolidation after the deadline reassignment, so it outperforms the PHGS in most cases. Moreover, the GAINM chooses the instance with the highest cost-performance to schedule tasks, and thus it is more suitable for scheduling multiple SWs. Because execution instances receive more tasks as the number of SWs rises, and thus their utilization



(a) Steady-load mode.



(b) Growing-load mode.

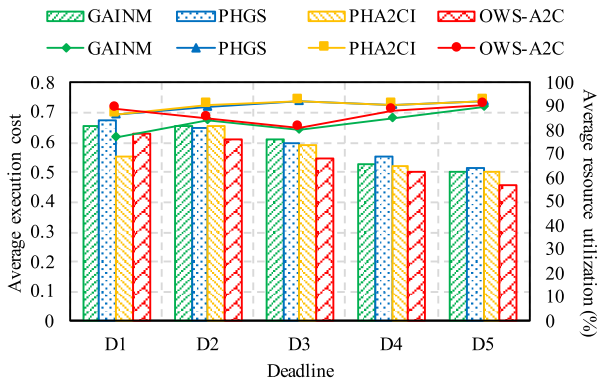


(c) Switching-load mode.

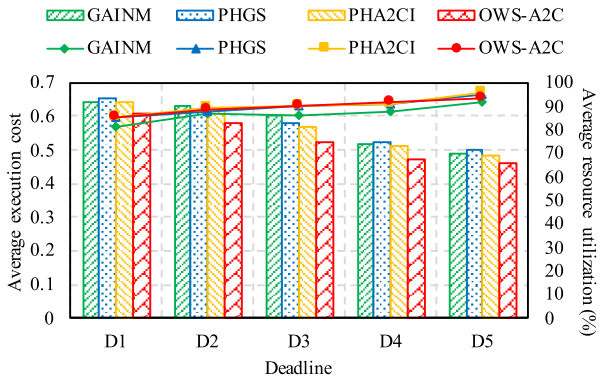
FIGURE 6. Performance comparisons of scheduling CyberShake under various load modes among different methods.

will increase and execution costs will decrease. But in some cases of the LIGO, the performance of the GAINM may be slightly inferior to the PHGS, as shown in Figures 8(a) and (b). This is because there are many parallel tasks in the LIGO, which might influence the optimization effect on execution costs when the deadline is tight (e.g. D1).

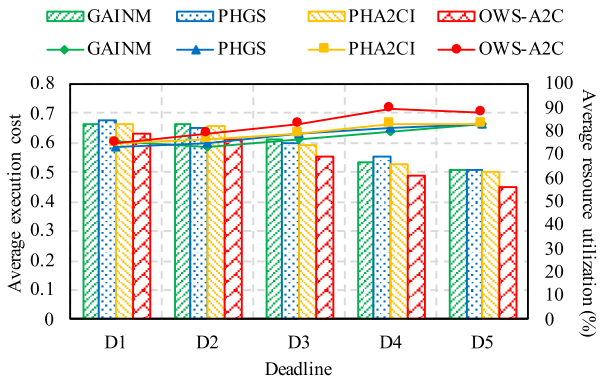
Moreover, as shown in Figures 7(b) and (c), when the deadline is tight (e.g. D1), the performance gap between



(a) Steady-load mode.

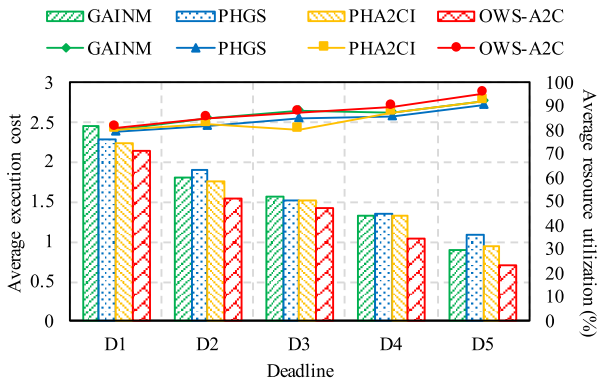


(b) Growing-load mode.

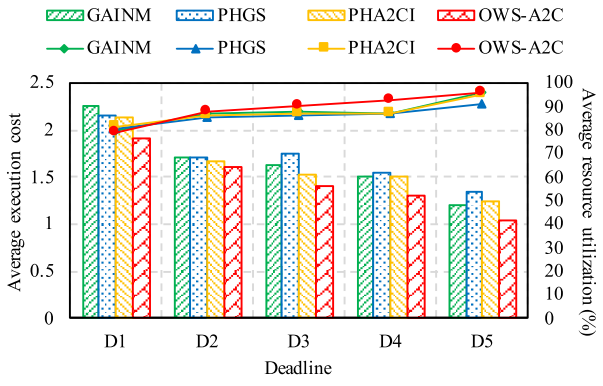


(c) Switching-load mode.

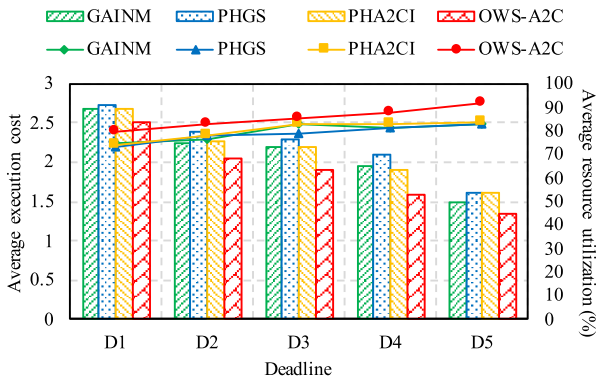
FIGURE 7. Performance comparisons of scheduling Epigenomics under various load modes among different methods.



(a) Steady-load mode.



(b) Growing-load mode.



(c) Switching-load mode.

FIGURE 8. Performance comparisons of scheduling LIGO under various load modes among different methods.

different methods is small. This is because all these methods tend to schedule tasks to the instance with the fastest processing speed when they face strict deadlines. Correspondingly, the average resource utilization of execution instances is not high in this case. Since there are many serial tasks in the Epigenomics, the optimization space for execution costs might thus be limited. With the relaxation of deadline constraints, the performance of the OWS-A2C becomes more excellent. This is because the proposed method comprehensively considers the cost-performance, resource utilization, and execution costs of execution instances in the process of scheduling optimization. By contrast, the PHGS does not take the above important factors into account, so its performance is severely degraded when deadline constraints become loose.

As for the other two types of SWs including Montage and SIPHT, their corresponding scheduling performances

achieved by using different scheduling methods under various load modes are similar to the cases of the CyberShake and Epigenomics, as shown in Figures 6 and 7, respectively. Therefore, the descriptions will not be repeated.

VI. CONCLUSION AND FUTURE WORK

The complex task dependencies and various resource cost-performance impose great challenges on scheduling multiple SWs with deadline constraints in MCEs. To address these challenges, we propose an OWS-A2C method to perform the online SW scheduling with adaptive resource allocation and consolidation. The extensive simulation experiments demonstrate the effectiveness of the proposed method in achieving adaptive and efficient SW scheduling in MCEs. More specifically, the OWS-A2C outperforms three baseline scheduling methods, including GAINM, PHGS, and PHA2CI, in terms of average execution cost and resource utilization. In future work, we will consider the interference of fluctuations of instances' execution performance on scheduling results and further improve the robustness of the proposed algorithm. Moreover, we will take the data transmission between tasks into account for extending the application scenarios of the algorithm and verify it in real-world environments.

ACKNOWLEDGMENT

(Zheyi Chen and Kai Lin contributed equally to this work.)

REFERENCES

[1] J. Fabra, M. J. Ibanez, P. Alvarez, and J. Ezpeleta, "Behavioral analysis of scientific workflows with semantic information," *IEEE Access*, vol. 6, pp. 66030–66046, 2018.

[2] B. Zhang, L. Yu, Y. Feng, L. Liu, and S. Zhao, "Application of workflow technology for big data analysis service," *Appl. Sci.*, vol. 8, no. 4, p. 591, Apr. 2018.

[3] B. Lin, F. Zhu, J. Zhang, J. Chen, X. Chen, N. N. Xiong, and J. L. Mauri, "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4254–4265, Jul. 2019.

[4] M. Sardaraz and M. Tahir, "A hybrid algorithm for scheduling scientific workflows in cloud computing," *IEEE Access*, vol. 7, pp. 186137–186146, 2019.

[5] W. Song, F. Chen, H.-A. Jacobsen, X. Xia, C. Ye, and X. Ma, "Scientific workflow mining in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2979–2992, Oct. 2017.

[6] M. Dickinson, S. Debroy, P. Calyam, S. Valluripally, Y. Zhang, R. B. Antequera, T. Joshi, T. White, and D. Xu, "Multi-cloud performance and security driven federated workflow management," *IEEE Trans. Cloud Comput.*, early access, Jun. 22, 2018, doi: 10.1109/TCC.2018.2849699.

[7] H. Hu, Z. Li, H. Hu, J. Chen, J. Ge, C. Li, and V. Chang, "Multi-objective scheduling for scientific workflow in multicloud environment," *J. Netw. Comput. Appl.*, vol. 114, pp. 108–122, Jul. 2018.

[8] M. Farid, R. Latip, M. Hussin, and N. A. W. Abdul Hamid, "Scheduling scientific workflow using multi-objective algorithm with fuzzy resource utilization in multi-cloud environment," *IEEE Access*, vol. 8, pp. 24309–24322, 2020.

[9] M. Masdari and M. Zangakani, "Efficient task and workflow scheduling in inter-cloud environments: Challenges and opportunities," *J. Supercomput.*, vol. 76, no. 1, pp. 499–535, Jan. 2020.

[10] M. Zotkiewicz, M. Guzek, D. Kliazovich, and P. Bouvry, "Minimum dependencies energy-efficient scheduling in data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 12, pp. 3561–3574, Dec. 2016.

[11] F. Xu, H. Zheng, H. Jiang, W. Shao, H. Liu, and Z. Zhou, "Cost-effective cloud server provisioning for predictable performance of big data analytics," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1036–1051, May 2019.

[12] V. Singh, I. Gupta, and P. K. Jana, "A novel cost-efficient approach for deadline-constrained workflow scheduling by dynamic provisioning of resources," *Future Gener. Comput. Syst.*, vol. 79, pp. 95–110, Feb. 2018.

[13] X. Li, J. Xu, and Y. Yang, "A chaotic particle swarm optimization-based heuristic for market-oriented task-level scheduling in cloud workflow systems," *Comput. Intell. Neurosci.*, vol. 2015, no. 1, pp. 1–11, 2015.

[14] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, "Scheduling workflows with budget constraints," in *Integrated Research in GRID Computing*. Boston, MA, USA: Springer, 2007, pp. 189–202.

[15] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 158–169, Jan. 2013.

[16] J. J. Durillo, V. Nae, and R. Prodan, "Multi-objective workflow scheduling: An analysis of the energy efficiency and makespan tradeoff," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput.*, May 2013, pp. 203–210.

[17] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. V. Hemert, "Scientific workflows: Moving across paradigms," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 1–39, Feb. 2017.

[18] B. Lin, W. Guo, N. Xiong, G. Chen, A. V. Vasilakos, and H. Zhang, "A pretreatment workflow scheduling approach for big data applications in multicloud environments," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 581–594, Sep. 2016.

[19] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," *Future Gener. Comput. Syst.*, vol. 48, pp. 1–18, Jul. 2015.

[20] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, and N. Najjari, "Online multi-workflow scheduling under uncertain task execution time in IaaS clouds," *IEEE Trans. Cloud Comput.*, early access, Mar. 19, 2019, doi: 10.1109/TCC.2019.2906300.

[21] B. Lin, W. Guo, and X. Lin, "Online optimization scheduling for scientific workflows with deadline constraint on hybrid clouds," *Concurrency Comput., Pract. Exper.*, vol. 28, no. 11, pp. 3079–3095, Aug. 2016.

[22] Y. Wang, Y. Guo, Z. Guo, T. Baker, and W. Liu, "CLOSURE: A cloud scientific workflow scheduling algorithm based on attack–defense game model," *Future Gener. Comput. Syst.*, vol. 111, pp. 460–474, Oct. 2020.

[23] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget and deadline aware e-Science workflow scheduling in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 29–44, Jan. 2019.

[24] B. Lin, W. Guo, and G. Chen, "Scheduling strategy for science workflow with deadline constraint on multi-Cloud," *J. Commun.*, vol. 39, no. 1, pp. 56–69, Jan. 2018.

[25] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *J. Grid Comput.*, vol. 12, no. 4, pp. 559–592, Dec. 2014.

[26] H. Kang, J. Koh, Y. Kim, and J. Hahm, "A SLA driven VM auto-scaling method in hybrid cloud environment," in *Proc. 15th Asia–Pacific Netw. Oper. Manage. Symp.*, Sep. 2013, pp. 1–6.

[27] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2011, pp. 1–12.

[28] P. Wangsom, K. Lavangnananda, and P. Bouvry, "Multi-objective scientific-workflow scheduling with data movement awareness in cloud," *IEEE Access*, vol. 7, pp. 177063–177081, 2019.

[29] J. Kumar and A. K. Singh, "Workload prediction in cloud using artificial neural network and adaptive differential evolution," *Future Gener. Comput. Syst.*, vol. 81, pp. 41–52, Apr. 2018.

[30] Z. Chen, J. Hu, G. Min, A. Y. Zomaya, and T. El-Ghazawi, "Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 923–934, Apr. 2020.

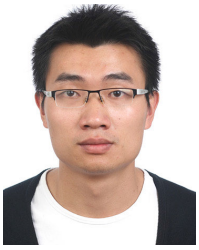
[31] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *Proc. 1st Int. Conf. E-Sci. Grid Comput. (E-Sci.)*, Jul. 2005, pp. 140–147.

[32] F. M. Chiussi and V. Sivaraman, "Achieving high utilization in guaranteed services networks using early-deadline-first scheduling," in *Proc. 6th Int. Workshop Qual. Service*, May 1998, pp. 209–217.

[33] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Proc. 3rd Workshop Workflows Support Large-Scale Sci.*, Nov. 2008, pp. 1–10.
[34] F. Pop, C. Dobre, V. Cristea, and N. Bessis, "Scheduling of sporadic tasks with deadline constrains in cloud environments," in *Proc. IEEE 27th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Mar. 2013, pp. 764–771.
[35] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 423–430.
[36] R. F. Da Silva. (2020). *Workflow Generator*. [Online]. Available: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowHub
[37] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

ZHEYI CHEN received the B.Sc. degree in computer science from Shanxi University, China, in 2014, and the M.Sc. degree in computer science from Tsinghua University, China, in 2017. He is currently pursuing the Ph.D. degree in computer science with the University of Exeter, U.K. His research interests include cloud computing, mobile edge computing, deep learning, reinforcement learning, and resource optimization.

KAI LIN received the B.S. degree in software engineering from Fujian Agriculture and Forestry University, China, in 2015. He is currently pursuing the M.S. degree in materials engineering with the College of Physics and Energy, Fujian Normal University, China. His current research interests include vehicular edge computing and cloud computing.



BING LIN received the B.S. and M.S. degrees in computer science and the Ph.D. degree in communication and information systems from Fuzhou University, Fuzhou, China, in 2010, 2013, and 2016, respectively. He is currently an Assistant Professor with the College of Physics and Energy, Fujian Normal University, Fujian, China. He is also the Academic Secretary of CCF YOCSEF in Fuzhou. He has authored or coauthored over 20 journals and conference papers, such as IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, and *Concurrency and Computation: Practice and Experience*. His research interests include parallel and distributed computing, computational intelligence, and data center resource management.

XING CHEN received the B.S. and Ph.D. degrees in computer software and theory from Peking University, Beijing, China, in 2008 and 2013, respectively. He is currently an Associate Professor and the Deputy Director of the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, and leads the Systems Research Group. He has authored or coauthored more than 30 journal and conference papers. His research interests include the software systems and engineering approaches for cloud and mobility. His current projects cover the topics from self-adaptive software, computation offloading, model-driven approach, and so on. He was a recipient of the first Provincial Scientific and Technological Progress Award in 2018.

XIANGHAN ZHENG received the M.Sc. degree in distributed system and the Ph.D. degree in information communication technology from the University of Agder, Norway, in 2007 and 2011, respectively. He is currently a Professor with the College of Mathematics and Computer Sciences, Fuzhou University, China. His current research interests include new generation networks with a special focus on cloud computing services and applications, and big data processing and security.

CHUNMING RONG (Senior Member, IEEE) is currently a Professor and the Head of the Center for IP-based Service Innovation (CIPSI), University of Stavanger (UiS), Norway. He has supervised 26 PhDs, nine PostDocs, and more than 60 master projects. He has extensive experience in managing large-scale Research and Development projects, both in Norway and EU. His research interests include cloud computing, data analytics, cyber security, and blockchain. He has been honored as a member of the Norwegian Academy of Technological Sciences (NTVA) since 2011. He has extensive contact network and projects in both the industry and academic. He is also a Founder and a Steering Chair of IEEE CloudCom conference and workshop series. He is also the Chair of IEEE CLOUD COMPUTING, an Executive Member of Technical Consortium on High Performance Computing (TCHPC), and the Chair of STC on Blockchain in IEEE Computer Society, and has served as a Global Co-Chair for IEEE Blockchain in 2018. He is also an Advisor of the StandICT.EU to support European scandalization activities in ICT. He is also the Co-Founder of two start-ups bitYoga and Dataunitor in Norway, both received EU Seal of Excellence Award in 2018. He was an Adjunct Senior Scientist leading Big-Data Initiative at NORCE from 2016 to 2019, and the Vice President of CSA Norway Chapter from 2016 to 2017. He is also a Co-Editor-in-Chief of the *Journal of Cloud Computing* Springer. He has also served as the Steering Chair from 2016 to 2019. He has also been serving a Steering Member and an Associate Editor for the IEEE TRANSACTIONS ON CLOUD COMPUTING (TCC) since 2016.