

© 2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

M. J. Nadjafi-Arani, S. Doostali and M. Younis, "Workflow Scheduling with Guaranteed Responsiveness and Minimal Cost," in *IEEE Transactions on Services Computing*, 2022, doi: 10.1109/TSC.2022.3222098.

<https://doi.org/10.1109/TSC.2022.3222098>

Access to this work was provided by the University of Maryland, Baltimore County (UMBC) ScholarWorks@UMBC digital repository on the Maryland Shared Open Access (MD-SOAR) platform.

**Please provide feedback**

Please support the ScholarWorks@UMBC repository by emailing [scholarworks-group@umbc.edu](mailto:scholarworks-group@umbc.edu) and telling us what having access to this work means to you and why it's important to you. Thank you.

# Workflow Scheduling with Guaranteed Responsiveness and Minimal Cost

Mohammad Javad Nadjafi-Arani, Saeed Doostali, Mohamed Younis, Fellow, IEEE

**Abstract**—Workflow scheduling is the process of optimally assigning virtual machines to workflow tasks subject to response time and cost consideration. Since such an optimization problem is NP-complete, providing an effective heuristic approach is essential. In this paper, we consider the workflow scheduling problem with the least cost subject to a bound on the response time. We show that existing solutions fundamentally care for the longest execution path within the workflow without appropriately handling non-critical paths. To overcome such a shortcoming, we propose a novel heuristic algorithm based on discrete mathematics. We first demonstrate that a workflow has a bijective relation with a partially ordered set and then introduce two operations on the workflow to show that it is an algebraic structure. We then form a totally ordered set,  $(T^{exe}(\mathcal{P}), \preceq)$ , of workflow paths where  $T^{exe}(\mathcal{P})$  is the set of path execution times. Based on  $(T^{exe}(\mathcal{P}), \preceq)$ , we identify the path with the maximum cumulative execution time and allocate a virtual machine to each task of the path based on the workflow deadline. We then delete the scheduled path and run our algorithm for each resulting sub-workflow in parallel. The results indicate that the proposed algorithm outperforms the best-known approaches in the literature.

**Keywords**—Workflow scheduling; partially ordered set; Cloud computing; Graph; Critical path; Heuristic algorithm.

## 1 INTRODUCTION

A workflow reflects a formal representation of a sequence of steps or tasks and is commonly used in a wide range of scientific applications in various fields such as bioinformatics, astronomy, and physics [1]. In recent years, cloud computing has attracted a great deal of attention as an effective means for providing an elastic computing infrastructure for handling large workflows [2], [3]. Workflow scheduling refers to the problem of mapping each of its tasks to a proper virtual machine (VM) in order to satisfy some performance goals. Unsurprisingly, faster computing services involve increased or more capable resources and consequently higher service fees. Therefore, scheduling faces a trade-off between service quality and cost. Since workflow scheduling is an NP-complete problem [4], providing an effective heuristic approach is necessary. On the other hand, in applications that must complete within a specific duration (deadline), missing a deadline may have catastrophic consequences. Target tracking, smart manufacturing and autonomous navigation systems are examples of these applications, in which the deadline constitutes a hard constraint that must be satisfied by the workflow scheduling algorithm. In addition, these applications involve extensive computation and increased concurrency for which cloud resources would avail. Intuitively, the deadline must exceed the workflow completion time based on the fastest schedul-

ing arrangement (i.e., scheduling all tasks on the fastest virtual machine) which is deemed as a feasibility condition.

A Directed Acyclic Graph (DAG) is conventionally used to model a workflow, where the graph nodes denote the computational tasks and the edges capture the dependencies among tasks. Most workflow scheduling algorithms found in the literature assume a DAG as the underlying representation. A common theme among published algorithms is to focus on the longest execution path within the workflow, which is often referred to as the critical path. Clearly, growing the execution latency on the critical path would extend the completion time (makespan) of the entire workflow, and would thus miss the deadline. Apart from scheduling the critical path, an important question is how to deal with uncritical paths; specifically, which path on the DAG must be considered next. As discussed Section 3.2, scheduling the wrong path can lead to deadline failure. Existing DAG based schedulers usually rank paths based on criticality and iterate over them. We show in Section 3.2 that such an approach does not guarantee meeting the scheduling objectives. In summary, published techniques care for the critical path at the expense of the optimization of non-critical tasks, which could limit the achievable performance or risk missing some hard deadlines.

This paper opts to overcome the aforementioned shortcomings. It first presents a bijective relation between workflows and algebraic structures. This relation helps in forming a totally ordered set,  $\mathcal{X}$ , of workflow paths. Then, we use such an ordered set to propose a novel heuristic algorithm for workflow scheduling on the cloud with minimum cost subject to a bound on the response time (i.e., deadline). We call our algorithm Workflow Scheduling with Guaranteed Responsiveness and Minimal Cost (GuRMiC). The set  $\mathcal{X}$  is used to determine the order of scheduling the workflow paths. Our algorithm specifies the start and finish times of tasks of the critical path by considering a suitable virtual

- M. J. Nadjafi-Arani is with the Faculty of Science, Mahallat Institute of Higher Education, Mahallat, Iran.  
Email: mjnadjafiarani@mahallat.ac.ir
- S. Doostali is with the Dept. of computer, Univ. of Kashan, Kashan, Iran, and the Faculty of Engineering, Mahallat Institute of Higher Education, Mahallat, Iran.  
Email: doostali@grad.kashanu.ac.ir
- M. Younis is with the Dept. of Computer Science and Elect. Eng., University of Maryland Baltimore County, Baltimore, MD 21250, USA.  
Email: younis@umbc.edu
- S. doostali is the corresponding author.

machine for them. GuRMiC then removes such a critical path from the given workflow, which yields some sub-workflows. Now, the algorithm is applied in each sub-workflow, where  $\mathcal{X}$  is used to pick the next path for scheduling within the sub-workflows. Hence, the set  $\mathcal{X}$  provides a total view of the main workflow when the algorithm tries to schedule the sub-workflows. In other words, the totally ordered set helps GuRMiC to consider the main workflow and sub-workflow simultaneously. We also propose an effective method for updating the allocated virtual machines by distributing the laxity (extra time) among the tasks to fix their start and finish times so that the overall execution cost is minimized. The effectiveness of GuRMiC is evaluated through simulation. The results show that GuRMiC outperforms the state-of-the-art methods in terms of the cost of executing workflows. Such distinct performance is further validated by utilizing the ANOVA and Tukey HSD statistical tests.

This paper is organized as follows. In Section 2, the scheduling system model will be presented. The problem is formulated and analyzed in Section 3. The proposed GuRMiC approach consists of two stages that are detailed in Sections 4, and 5, respectively. The performance of GuRMiC is evaluated through simulation in Section 6. Section 7 reviews published work of constrained cloud-based workflow scheduling. Finally, the paper is concluded in Section 8.

## 2 SYSTEM MODEL

### 2.1 Workflow Model

A workflow is modeled by a directed acyclic graph  $G = (V, E)$  in which  $V$  is the set of  $n$  computational tasks (i.e.,  $V = \{t_1, t_2, \dots, t_n\}$ ) and  $E$  is the set of control and data dependencies between tasks which are indicated by directional edges. An edge  $(t_i, t_j) \in E$  represents a precedent constraint, such that the execution of task  $t_j$  can start only after completing the execution of  $t_i$  with all data received from it. Hence, we can define  $t_i$  as the parent of  $t_j$  (denoted by  $t_i \in \text{Parent}(t_j)$ ), and consequently  $t_j$  as the child of  $t_i$  (i.e.,  $t_j \in \text{Child}(t_i)$ ). In general, a task can start when all of its parents have been executed, and the required data have been received from its parents. In a given graph, a task with no parent is known as an entry task, and a task with no child is known as an exit task. Without loss of generality, we assume these two tasks (i.e., entry and exit tasks) are unique in the graph. Clearly, if there is more than one entry (exit) task, then we can add a virtual task  $t_{\text{entry}}$  ( $t_{\text{exit}}$ ) to the beginning (end) of the workflow. These tasks have zero execution time, and are connected to real entry and exit tasks by edges with zero data transmission time. In other words, it is assumed that a workflow constitutes a connected DAG.

### 2.2 Cloud Model

Our cloud model consists of an Infrastructure-as-a-service (IaaS) provider which offers virtual machines as computational resources to execute workflows. The service provider offers computation services like Amazon EC2 (Amazon Elastic Compute Cloud [5]), where instances are provisioned on demand. Moreover, our basic pricing model is a pay-as-you-go basis, i.e., the service provider charges users based

on the number of time intervals that they have used. Note that if users utilize only a fraction of the last interval, they must pay for the whole interval. In fact, IaaS cloud providers consider multiple virtual machine types containing various amounts of resources with different costs per use, and each virtual machine  $vm_k$  is associated with a virtual machine type.

There are two basic assumptions about resources in the cloud: (a) resources are abundant, and (b) resources are heterogeneous. Assumption (a) says that the resource pool in the cloud is almost unlimited, and users can access resources at any time and as much as they want, while assumption (b) states that there are different instance types in the cloud. The type associated with a set of resources includes various Quality of Service (QoS) features such as CPU type and memory size and different prices. This means that a resource with a faster CPU or more memory requires a higher investment, and users have to pay more to use it. We also assume that all computation and storage services are located in the same physical region. Note that in most real clouds, such as Amazon, the internal data transfer is free, so we do not consider any cost for data transmission in our approach. Clearly, the service provider charges the users for using the storage service, but it has no effect on the proposed scheduling method.

### 2.3 Problem Statement

The workflow scheduling problem is fundamentally a resource allocation optimization that arises in many practical applications. In the context of cloud infrastructure, tasks are to be assigned computational resources subject to constraints. For GuRMiC, the constraints are mainly precedence and timing constraints, where tasks execution sequence is governed by a certain flow, while the time constraint reflects the maximum allowed time for completing all tasks. Both types of constraints are mandated by the application. Such workflow scheduling problem is known to be NP-hard. An example application scenario is in manufacturing where a cyber-physical system (CPS) is employed to coordinate components fabrication and assembly. Such a CPS is composed of multiple control modules, each is to be activated in a specific order, and the entire processing of a single manufactured item has to complete within an allotted time according to the production pipeline.

There are multiple variants of workflow scheduling [6]. Such variation is derived by the application scenario and affects the problem formulation and solution strategy. In general, an optimization problem can be categorized by the objective function and constraints. The workflow scheduling on cloud infrastructure can be classified according to the objective function as: (i) single objective, and (ii) multi-objective. The objective function itself can be linear and nonlinear. The workflow scheduling can also be either constrained or unconstrained. An example of the latter is when there is no inter-task dependency and minimizing the cost and/or execution latency is the only concern. GuRMiC is geared to solve a variant of the problem where a linear cost function is to be minimized subject to task dependence and workflow completion constraints. Such a problem is popular in practice in industrial, transportation, and defense applications that involve autonomous control.

### 3 SCHEDULING OPTIMIZATION AND ANALYSIS

In this section we formulate the workflow scheduling as an optimization problem and motivate our discrete mathematics solution strategy by highlighting the shortcomings of existing DAG based schemes using an example.

#### 3.1 Problem Formulation

GuRMiC tackles the execution time and cost trade-off in the workflow scheduling. To formulate the scheduling optimization problem, we first define the execution time of each task of the workflow, then propose a relation for makespan, and finally compute the cost based on the popular IaaS charging model. Let  $T_{i,k}$  be the execution time of a task  $t_i$  on a virtual machine  $vm_k$  of a certain type. Accordingly, the finish time of task  $t_i$ ,  $F(t_i)$ , is defined as follows:

$$F(t_i) = S(t_i) + T_{i,k}, \quad (1)$$

where  $S(t_i)$  is the start time of  $t_i$  which is calculated by:

$$S(t_i) = \max_{t_j \in \text{Parent}(t_i)} F(t_j). \quad (2)$$

This equation states that the start time of each task is equal to the maximum finish time of its parents. Note that the start time of  $t_{\text{entry}}$  is zero because it is the entry task of the workflow and has no parent. The total execution time of a workflow (i.e., makespan) corresponds to the finish time of  $t_{\text{exit}}$  since it is the last task that can be executed. Hence, the makespan of a given workflow is defined as:

$$\text{Makespan} = F(t_{\text{exit}}). \quad (3)$$

There are different pricing strategies in the IaaS platforms. Amazon EC2, for example, considers one-hour intervals, while Microsoft Azure allocates one-minute intervals to execute user tasks. Let  $l_{\text{int}}$  be the length of the time interval based on the pricing strategy. Hence, the cost of executing task  $t_i$  can be defined as follows:

$$C_i = \text{Cost}_k \times \left\lceil \frac{T_{i,k}}{l_{\text{int}}} \right\rceil, \quad (4)$$

where  $\text{Cost}_k$  is the cost of virtual machine  $vm_k$  for each time interval. Hence, the overall cost of executing all tasks is

$$\text{Cost} = \sum_{t_i \in V} C_i. \quad (5)$$

Generally, the cost is an important factor for the users, where they prefer to use cheaper services with less QoS as long as their needs and expectations are met. Hence, the users define a deadline for when they expect the workflow to be completely handled. In this paper, we try to find proper scheduling for workflows with a minimum execution cost while satisfying the user-defined deadline.

#### 3.2 Limitation of DAG-based Schedulers

This subsection highlights the main shortcoming of existing DAG-based approaches that focus on the critical path without appropriately handling non-critical paths. To schedule a critical path,  $\eta$ , existing approaches determine the start and finish times of each task on  $\eta$  such that the last task completes execution before the deadline. To optimize the

cost, tasks are assigned to the least expensive virtual machine, and hence the finish time could be pushed. Such task completion latency is deemed acceptable as long as the last task on  $\eta$  does not violate the deadline. However, in some workflows, some non-critical paths with the same prefix sub-path could fork from and then join  $\eta$ . Assume that  $\zeta_1$  and  $\zeta_2$  are two such paths, and let us schedule  $\zeta_1$  before  $\zeta_2$ . In such a case, inter-path dependency could nullify the validity of the critical path schedule. The reason is that instrumenting latency within  $\zeta_1$  to reduce cost will delay the start of the postfix sub-path of  $\zeta_2$  and consequently increase the completion time of the task at the joining point with  $\eta$ , i.e., the last task of  $\zeta_2$ .

To illustrate, let us consider the sample workflow of Fig. 1 and assume that there exist three virtual machines,  $vm_1$ ,  $vm_2$ , and  $vm_3$ , available for executing such workflow. The required processing time of a task on  $vm_2$  and  $vm_3$  is 2 and 3 times that of  $vm_1$ , respectively. Let 8, 7, 4, 1, 7, 2, 1, 3, 6, 9 be the execution times of tasks  $t_1$  to  $t_{10}$  on  $vm_1$ . Moreover, assume that the cost of each time interval is equal to 5 for  $vm_1$ , 3 for  $vm_2$ , and 2 for  $vm_3$ , and the overall deadline of the workflow is 32. Note that the deadline is a hard constraint and must be satisfied by scheduling algorithms.

According to the task execution times, path  $P_1: t_{\text{entry}}-t_2-t_5-t_8-t_9-t_{10}-t_{\text{exit}}$  is the critical path of the workflow. The given deadline requires assigning the tasks on this path to  $vm_1$ . According to such a schedule, task  $t_4$  must be started after 14 and task  $t_7$  must be done before 17. Now, we have three sub-paths,  $P_2: t_4-t_7$ ,  $P_3: t_4-t_6$ , and  $P_4: t_1-t_3-t_6$ . Let us consider IC-PCP [7], which is one of the most cited DAG-based scheduling algorithms in the literature. IC-PCP selects path  $P_3$  and schedules it on  $vm_3$ . Such allocation of  $P_3$  forces task  $t_7$  to start after 17, which causes task  $t_7$  to be unschedulable. It may come to mind that selecting a path that connects to an earlier task in the critical path can be a solution; however, a similar example can easily show that this solution also suffers from the same problem. Hence in this paper, we propose a novel heuristic algorithm based on discrete mathematics to overcome this shortcoming.

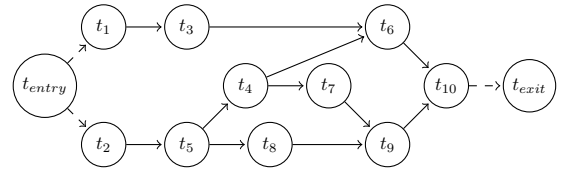


Fig. 1. A simple workflow with 10 tasks

### 4 DETAILED GURMIC DESIGN

The proposed GuRMiC algorithm schedules workflows based on QoS and a totally ordered set of paths with the longest execution time. It tries to minimize the execution cost of a workflow while satisfying the user-defined deadline. GuRMiC consists of two stages. First, the execution time of the various paths is calculated and the critical path is determined. For such a stage, some algebraic properties are leveraged. The first stage is the focus on this section. The second stage is geared for allocating virtual machines to achieve the optimization objective while meeting the constraints. The second stage is detailed in the next section.



#### 4.1 Relating Workflows to Discrete Structures

This section first covers some preliminaries for relations for sets and then presents a bijection function between workflows and algebraic structures. The used notation is based on [8], which can be further referred by interested readers for more discussion of fundamental concepts.

Let  $Q$  be a non-empty set, and  $R \subseteq Q \times Q$ .  $R$  is a relation on the set  $Q$ , if  $(a, b) \in R$ , it implies that  $a$  is in a relationship with  $b$ .  $R$  is called a partial order set whenever it is reflexive, antisymmetric and transitive. In mathematics, symbol  $\preceq$  is used for any arbitrary order relation  $R$  on a set  $Q$ . When this symbol models a typical ordinal position of numbers, we say  $(Q, \preceq)$  is a partially ordered set (poset). If  $(x, y) \in R$ , then we write  $x \preceq y$  indicating that  $y$  is the successor of  $x$  or  $x$  is the antecedent of  $y$ . We write  $x \prec y$  when there is no element such as  $z$  between  $x$  and  $y$ . In other words, an element such as  $z \in Q$  cannot be found such that  $x \prec z \prec y$  (If  $a \preceq b$  and  $a \neq b$ , then we usually write  $a \prec b$ ). An element  $a \in Q$  is called maximal when it is not smaller than any other member of  $Q$ , and we call it the maximum when  $a$  is larger than all the members of  $Q$ . Similarly, element  $b \in Q$  is called minimal when no other member of  $Q$  is less than it, and we call it the minimum when it is smaller than all the members of  $Q$ . A total order or linear order is a partial sequence on a set whose all members are related to each other. A totally ordered set is also called a chain.

Now consider a workflow that is represented as a finite acyclic graph, in which all edges are one-way flows from the sources to the sinks. It can be easily shown that the set of tasks (nodes) of a workflow,  $W$ , under the node processing relation is a poset. In other words, if  $u$  and  $v$  are two distinct arbitrary nodes of  $W$ , the expression  $u \preceq v$  represents a precedence constraint which indicates that the execution of task  $u$  should be completed before task  $v$  can start. Moreover, we assume that each node has a precedence constraint of itself. These assumptions satisfy the reflexive, antisymmetric, and transitive properties of a poset. These concepts are utilized in presenting our GuRMic approach.

#### 4.2 Path Execution Time Estimation

Based on the workflow scheduling goals, each task of a workflow must be assigned to a virtual machine that has the least cost while the completion time constraint (i.e., deadline) is satisfied. Assume that there are  $m$  available virtual machines,  $vm_1, \dots, vm_m$ , with fixed data processing power. Specifically, the relative processing power of  $vm_s$  to  $vm_k$  is a constant number, denoted by  $c_{sk}$ . Although the cost of faster virtual machines is always more than the slower ones, we assume that the cost ratio is still fixed, and matches the processing power, i.e., the relative cost is also  $c_{sk}$ . To reduce the overall cost, it is necessary to schedule the tasks of each separate sub-workflow on the slowest possible virtual machine so that the deadline is not violated. To this end, we must specify the start and finish times of each task so that it is not possible to assign these tasks to a slower machine. It is clear that such an assignment can minimize the overall execution cost.

Let us assume a workflow with  $n$  tasks and a given deadline. As shown in Section 4.1, a workflow is a poset that contains maximal chains. Note that each maximal chain

is a path starting from a minimal and ending at a maximal element. Let  $\mathcal{P} = \{P_1, \dots, P_r\}$  be the set of maximal chains and  $t_1^{(T)}, t_2^{(T)}, \dots, t_r^{(T)}$  be the elements of chain  $P_T$ . GuRMic computes the execution time of the chain  $P_T$  as follows. Let  $S(t_q^{(T)})$  be the start time of element  $t_q^{(T)}$  and  $F(t_q^{(T)})$  be the finish time of such an element. Based on the definitions of Section 3.1,  $F(t_q^{(T)})$  is equal to the start time of element  $t_q^{(T)}$  plus the time it takes to execute on a fixed virtual machine  $vm_j$ , while  $S(t_q^{(T)})$  is equal to the maximum finish time of its parents. Note that if the minimal element in the poset is the successor node of  $t_{entry}$ , then the start time of this element is zero; otherwise its start time is equal to the maximum finish time of its antecedent elements. According to the above explanation, the execution time of chain  $P_T$ ,  $T^{exe}(P_T)$ , is defined as the finish time of its maximal element.

In a workflow, there could be paths that have the same execution time. For these paths, we define the longer relation. Let  $P_T$  and  $P_J$  be two paths with the same execution time; the longer relation is defined as  $T^{exe}(P_T) > T^{exe}(P_J)$  if, recursively from maximal to minimal, there is an element in  $P_T$  whose finish time is greater than the finish time of the equivalent element in chain  $P_J$ . If each node in  $P_T$  has the same execution time as its corresponding node in  $P_J$ , then we randomly pick one of these paths as the longer path. We illustrate this relationship with the simple example in Fig. 2. Let 1, 2, 5, 2, 10, 8 be the execution times of tasks  $t_1$  to  $t_6$  on a given virtual machine. Note that the execution times of  $t_{entry}$  and  $t_{exit}$  are zero. Based on these times and Eqs. (1) and (2), we compute the start and finish times of each node (see Table 1). For instance, the start times of  $t_1$  and  $t_2$  are zero because the finish time of their parent (i.e.,  $t_{entry}$ ) is zero, while the start time of  $t_3$  is 2, which is equal to the maximum finish time of its parents (i.e.,  $t_1$  and  $t_2$ ). Note that the finish time of each task is its start time plus its execution time. On the other hand since the execution time of the virtual task  $t_{exit}$  is zero, so its start time is equal to its finish time. As shown in Fig. 2, we have six paths:

- $P_1: t_{entry} - t_1 - t_5 - t_{exit}$
- $P_2: t_{entry} - t_1 - t_3 - t_4 - t_5 - t_{exit}$
- $P_3: t_{entry} - t_1 - t_3 - t_4 - t_6 - t_{exit}$
- $P_4: t_{entry} - t_2 - t_6 - t_{exit}$
- $P_5: t_{entry} - t_2 - t_3 - t_4 - t_5 - t_{exit}$
- $P_6: t_{entry} - t_2 - t_3 - t_4 - t_6 - t_{exit}$

It is clear that the execution time of all paths is equal to 17 (the finish time of the maximal element, i.e.,  $t_{exit}$ ). According to the longer relation, we need to compare the finish time of the parents of  $t_{exit}$ ; among its parents, task  $t_5$  has the longest finish time, so the execution time of paths  $P_1$ ,  $P_2$ , and  $P_5$  is greater than others. This process is repeated for task  $t_5$ , and among its parents (i.e.,  $t_1$  and  $t_4$ ), task  $t_4$  will be selected. It concludes that the execution time of  $P_2$  and  $P_5$  is greater than  $P_1$ . By repeating this process, path  $P_5$  is introduced as the first path with maximum execution time, and  $P_2$  is considered as the second one.

Let  $T^{exe}(\mathcal{P})$  be the set of execution times of the workflow paths in  $\mathcal{P}$ . The set  $T^{exe}(\mathcal{P})$  constitutes a poset under the longer relation (i.e.,  $\preceq$ ). Since we can compare the execution times of two arbitrary paths, all of the members of  $T^{exe}(\mathcal{P})$  are related to each other, and  $(T^{exe}(\mathcal{P}), \preceq)$  forms a totally

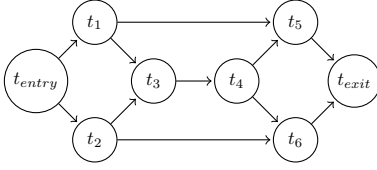


Fig. 2. Critical path example

TABLE 1  
Start and finish times for the tasks of Fig. 2

Time	Tasks							
	$t_{entry}$	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$	$t_{exit}$
Start	0	0	0	2	7	9	9	17
Finish	0	1	2	7	9	19	17	17

ordered set. The path corresponding to the maximum element of  $(T^{exe}(\mathcal{P}), \preceq)$  is called the critical path of a workflow. Note that  $(T^{exe}(\mathcal{P}), \preceq)$  is independent of selecting a virtual machine such as  $vm_k$  for a fixed  $1 \leq k \leq m$ , because virtual machines have different processing times under a constant coefficient (i.e.,  $c_{sk}$ ). Nonetheless, the critical path is not necessarily unique. To illustrate let us assume that  $t_1$  and  $t_2$  in Fig. 2 have the same execution time; in such a case both  $P_1$  and  $P_5$  qualify as the critical path. The definition of  $(T^{exe}(\mathcal{P}), \preceq)$  is the basis for our proposed algorithm for workflow scheduling as explained next.

**Remark 4.1.** In prominent competing approaches, such as IC-PCP and IC-PCPD2 [7] a critical path,  $P$ , is defined based on the fastest virtual machine,  $vm_f$ . Hence, scheduling  $P$ , on a slower virtual machine,  $vm_s$  implies that the ratio of the task execution times on these virtual machines is not constant, and consequently the critical path may be different for  $vm_s$ . In other words, prior work considers  $vm_f$  as a reference to find  $P$ , while using a virtual machine other than  $vm_f$  may yield  $P' \neq P$ .

## 5 SCHEDULING ALGORITHM

GuRMiC opts to minimize the execution cost while satisfying the workflow deadline. In the following, we present the details of our algorithm and provide an example to illustrate the steps. Algorithms 1 and 2 show the pseudo-code of the proposed method. First of all, the algorithm adds two virtual nodes  $t_{entry}$  and  $t_{exit}$  to the workflow if they do not exist. Then, it computes the start and finish times for each task and creates the poset  $(T^{exe}(\mathcal{P}), \preceq)$ , which constitutes the first stage, as we explained in Section 4. Since, in the worst case, the given workflow can be scheduled on the fastest virtual machine, we compute the initial value of start and finish times of each workflow task based on this machine. Next, the algorithm calls the *ScheduleTask* procedure, which schedules all workflow tasks and updates their start and finish times according to the slowest possible virtual machine, as detailed next.

### 5.1 Scheduling Procedure

In the *ScheduleTask* procedure (Algorithm 2), let  $P: t_1-t_2-\dots-t_j$  be the critical path of a given workflow. Clearly,  $t_1$  is the minimal element and  $t_j$  is the maximal element of

such a chain. Assuming all tasks of  $P$  (i.e., elements of the chain) are assigned to the slowest possible virtual machine, we estimate the total execution time and check the deadline violation. Scheduling the critical path specifies the start and finish times of each task  $t_i$  on this path. Such specification causes each node  $v$  in the workflow that is connected to a task  $t_i$  on  $P$  to have a start time constraint (if  $v$  is a successor of  $t_i$ ) or a finish time constraint (if  $v$  is an antecedent of  $t_i$ ). Note that if  $v$  has a deadline, any update in the finish time should not violate such a deadline. In GuRMiC, the deadline of a workflow is expected to be defined via:

$$Deadline = Makespan^{fast} \times \gamma, \quad (6)$$

where  $Makespan^{fast}$  is the makespan of the fastest schedule (i.e., the execution time when scheduling tasks on the fastest virtual machine), and  $\gamma \geq 1$  is the deadline factor. A tight deadline is defined by a smaller  $\gamma$  which forces the algorithm to select a faster virtual machine; meanwhile, a larger  $\gamma$  means a soft deadline which provides opportunities for selecting cheaper virtual machines. Thus, in the worst case (i.e., when the deadline is tight), we can schedule  $P$  on the fastest virtual machine.

After finding a suitable virtual machine for the selected path, the start and finish times of all elements of this chain will be specified. We define  $\alpha$  as the difference between the finish time of each node in the selected path and the minimum start time of its successor nodes (i.e., its child nodes). Note that the start time of the successor of the maximum node is equal to the workflow deadline. Such a deadline helps us to find  $\alpha$  for the maximal element of the selected chain, i.e.,  $t_j$ . The value of  $\alpha$  represents the time allowance that a task can leverage to reduce cost by using a slower virtual machine. The steps are as follows. We calculate  $\alpha$  for  $t_j$ , and then check whether it is possible to execute  $t_j$  on a slower machine (Line 9 of Algorithm 2). If not possible, we will increment the start and finish times of  $t_j$  by a value of  $\alpha$ . If the finish constraint (deadline),  $\beta$ , will be violated with such an increment, the start and finish time will be pushed forward by only  $\beta - F(t_j)$ , where  $F(t_j)$  is the finish time of task  $t_j$ . The same time adjustment logic is applied to other tasks of the selected path and their virtual machines are updated. The sequence is from maximal to minimal, i.e., from  $t_j$  to  $t_1$ .

Note that changing the start time of a task that has no unscheduled parent(s) delays the start of its child nodes and has a negative effect on the proposed algorithm. More details will be provided in Section 5.3. Hence, we examine the tasks of the selected path from the minimal node to the maximal one and shift back the start time of tasks by  $\alpha'$  until reaching the first task which has at least one unscheduled parent(s), where  $\alpha'$  is the difference between the start time of each node and the maximum finish time of its predecessors. Note that this process provides a minimum start constraint for the successor tasks.

Upon allocating the tasks of the critical path,  $P$ , GuRMiC handles the remaining tasks within the workflow as follows. First, the nodes of the workflow that are connected to  $P$  will have their start and finish times updated, as explained above. Next, we remove the nodes of path  $P$  from the workflow, which will partition the workflow into disjoint blocks

### Algorithm 1 Pseudo-code of the GuRMiC Algorithm

Procedure GuRMiC(Workflow  $W$ , Deadline  $D$ )

- 1: Add two virtual nodes  $t_{entry}$  and  $t_{exit}$  to  $W$
- 2: Compute  $S(t_i)$  and  $F(t_i)$  for each task of  $W$
- 3: Create the poset  $\mathcal{X} = (T^{exe}(\mathcal{P}), \preceq)$
- 4: ScheduleTask( $W, D, \mathcal{X}$ )

### Algorithm 2 Pseudo-code of the ScheduleTask procedure

Procedure ScheduleTask(Workflow  $W$ , Deadline  $D$ , Poset  $\mathcal{X}$ )

- 1: Create  $\mathcal{P}'$  as the set of maximal chains in  $W$
- 2:  $P \leftarrow$  the longest path of  $\mathcal{P}'$  based on poset  $\mathcal{X}$
- 3:  $vm_s \leftarrow$  the slowest possible virtual machine that can execute path  $P$ , i.e., does not violate the deadline,  $D$ , and also the start and finish constraints
- 4: **for each**  $t_i \in P$  **do**
- 5:     Update  $S(t_i)$  and  $F(t_i)$  based on  $vm_s$
- 6: **for each**  $t_i \in P$  **do**      $\triangleright$  from the maximal node to the minimal
- 7:     Compute the value of  $\alpha$
- 8:     **while** ( $\alpha \neq 0$ ) **do**
- 9:         **if** ( $S(t_i) + T_{i,k} \leq F(t_i) + \alpha$  and  $c_{sk} > 1$ ) **then**
- 10:             Update  $S(t_i)$  and  $F(t_i)$  based on  $vm_k$
- 11:             Compute the value of  $\alpha$
- 12:         **else**
- 13:             **if** ( $F(t_i) + \alpha > \beta$ ) **then**
- 14:                  $\alpha \leftarrow \beta - F(t_i)$
- 15:             Increase  $S(t_i)$  and  $F(t_i)$  by  $\alpha$
- 16: **for each**  $t_i \in P$  **do**      $\triangleright$  from the minimal node to the maximal
- 17:     **if** ( $t_i$  has at least one unscheduled parent) **then break**
- 18:     **else shift back**  $t_i$  by  $\alpha'$
- 19: **for each**  $t_i \in P$  **do**
- 20:     Compute the start constraints for all successors of  $t_i$
- 21:     Compute the finish constraints for all antecedents of  $t_i$
- 22: Remove the nodes of  $P$  from the workflow
- 23: **for each** block  $w \subset W$  **do in parallel**
- 24:     ScheduleTask( $w, D, \mathcal{X}$ )

(sub-workflows). GuRMiC then considers these blocks individually and repeats the steps. We note that the set of paths in a block  $G$  is a set of sub-paths of the original workflow  $W$ , which have already been characterized in the totally ordered set  $(T^{exe}(\mathcal{P}), \preceq)$ . After handing  $P$ , indeed GuRMiC removes the longest path of  $(T^{exe}(\mathcal{P}), \preceq)$ , and hence for  $G$ , it considers the second longest path in  $(T^{exe}(\mathcal{P}), \preceq)$ . Let  $P'$  be the second longest path. Now, if there is a sub-path of  $P'$  in a block, we select it as the longest path and repeat the aforementioned steps. Note that the proposed operations are repeated until the last path of  $(T^{exe}(\mathcal{P}), \preceq)$  is removed.

In summary, GuRMiC divides the workflow into several smaller blocks, and handles them individually. Thus, during the execution of the algorithm, the initial workflow is divided into smaller workflows, and new minimal and maximal elements are obtained. Note that the new minimal elements are not necessarily the successor of  $t_{entry}$ ; hence their start times are not zero, and each of them has a start constraint. There are similar conditions for the new maximal elements while they have finish constraints.

## 5.2 Illustrative Example

We illustrate the steps of the proposed approach with an example. Consider the graph of a sample workflow in Fig. 3. This graph consists of 16 tasks labeled with  $t_1$  to  $t_{16}$ , and two virtual tasks,  $t_{entry}$  and  $t_{exit}$ . Assume that three virtual machines  $vm_1$ ,  $vm_2$ , and  $vm_3$  are available to execute the workflow tasks; Table 2 shows the execution

time of each task in the workflow on these virtual machines. Also, assume that the cost of a time unit is 5 for  $vm_1$ , 3 for  $vm_2$ , and 2 for  $vm_3$ . Finally, let the overall deadline of the workflow be 31, which is a hard deadline corresponding to Eq. (6) with  $\gamma$  equals to one. Note that the execution times of tasks  $t_{entry}$  and  $t_{exit}$  are zero.

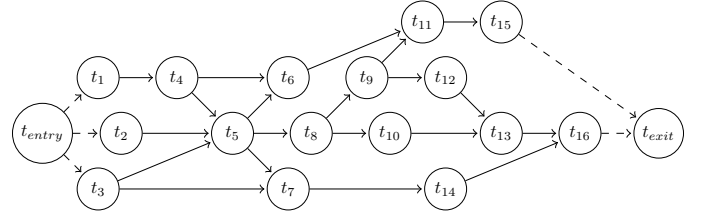


Fig. 3. An example workflow for illustrating the steps of GuRMiC

TABLE 2  
The execution times for the workflow of Fig. 3.

VM	Tasks															
	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$	$t_7$	$t_8$	$t_9$	$t_{10}$	$t_{11}$	$t_{12}$	$t_{13}$	$t_{14}$	$t_{15}$	$t_{16}$
$vm_1$	2	4	3	1	3	5	1	7	1	3	4	1	5	1	5	9
$vm_2$	4	8	6	2	6	10	2	14	2	6	8	2	10	2	10	18
$vm_3$	6	12	9	3	9	15	3	21	3	9	12	3	15	3	15	27

Based on the critical path definition and the data of Table 2, the chain  $P_1: t_{entry}-t_2-t_5-t_8-t_{10}-t_{13}-t_{16}-t_{exit}$  is the maximum element of poset  $(T^{exe}(\mathcal{P}), \preceq)$  (i.e., the critical path of the workflow). GuRMiC chooses  $vm_1$  as the cheapest virtual machine that can complete the execution of the tasks of  $P_1$  before the deadline. By assigning this virtual machine to the tasks of  $P_1$ , the start and finish times of these tasks are specified in row 1 of Table 3. In this table, each path corresponds to a row with two parts: top and bottom. The times of workflow tasks before and after scheduling the task of that row are demonstrated in the top and bottom parts, respectively. As stated below, the times of  $P_1$  tasks do not change; hence the bottom part of row 1 is ignored. The next step is to update the start and finish times of all unassigned successors and predecessors of the tasks of  $P_1$ . However, due to the tight deadline in this example, no update is made. For the same reason  $\alpha$  is set to zero, and the allocated virtual machines do not change. Now, we remove the nodes of  $P_1$  from the workflow, and obtain two sub-workflows, namely  $W_1$  and  $W_2$ , as shown in Fig. 4. Applying GuRMiC to these sub-workflows is subject to the following constraints, which are annotated in the figure next to the corresponding nodes.

- " $< 4$ " for tasks  $t_3$  and  $t_4$  (they must finish before  $t_5$ )
- " $> 7$ " for tasks  $t_6$  and  $t_7$  (they must start after  $t_5$ )
- " $> 14$ " for task  $t_9$
- " $< 17$ " for task  $t_{12}$
- " $< 22$ " for task  $t_{14}$

$W_2$  (the sub-workflow at the bottom of Fig. 4), has only one chain, referred to as  $P_2$ . According to the data in Table 2 and the stated constraints, the tasks of  $P_2$  can be scheduled on  $vm_1$ . Now, we calculate the value of  $\alpha$  for the maximal node of this chain, namely,  $t_{14}$ . In this case,  $\alpha$  is the difference between the start time of the child of  $t_{14}$ , i.e.,  $t_{16}$  which is denoted by the finish constraint, and the finish time of  $t_{14}$ , which is equal to 13. Based on the execution



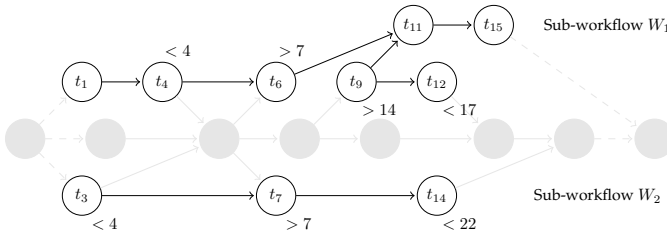


Fig. 4. Sub-workflows of Fig. 3; after removing the critical path

times and the value of  $\alpha$ , task  $t_{14}$  can be scheduled on  $vm_3$ . Next, we update the value of the extra time that is equal to 11. Since there is no slower machine than  $vm_3$  to schedule task  $t_{14}$  on, the new value of  $\alpha$  has no effect on changing the virtual machine of  $t_{14}$ ; hence the start and finish times of  $t_{14}$  are incremented by  $\alpha = 11$ . Clearly, such a change does not violate the finish constraint of  $t_{14}$ . We repeat this process for  $t_7$ , where according to the value of  $\alpha$ , it can be scheduled on  $vm_3$ . After that, the start and finish times of  $t_7$  are shifted forward by 9 (which is equal to the new value of  $\alpha$ ). For task  $t_3$ , the value of extra time is equal to 1 because task  $t_5$  introduces " $< 4$ " as a finish constraint for  $t_3$ . This constraint states that the execution of  $t_3$  must be completed before 4; hence the execution time of  $t_3$  can be extended by only one unit of time. Based on the data of Table 2, we are not able to change the virtual machine of  $t_3$  to a slower one; hence, we increase the start and finish times of  $t_3$  by 1. As we mentioned earlier, pushing forward the start time of a task that has no unscheduled parent(s), has a negative effect; thus, we shift back the start time of tasks  $t_3$ ,  $t_7$ , and  $t_{14}$  by the value of  $\alpha'$ . Note that according to Eq. (1), shifting back the start time of a task by  $\alpha'$  decreases its finish time by  $\alpha'$ . Hence, the start and finish times of task  $t_3$  are decreased by 1 and the times of tasks  $t_7$  and  $t_{14}$  are shifted back by 9. In fact, task  $t_{14}$  is shifted forward by 11, but scheduling  $t_7$  on  $vm_3$  decreases the value of  $\alpha'$  of  $t_{14}$  by 2. However, these adjustments affect the extra time that each task incurs. The above description is shown in row 2 of Table 3.

On the other hand,  $W_1$  has three sub-paths  $P_3$ :  $t_9$ - $t_{12}$ ,  $P_4$ :  $t_9$ - $t_{11}$ - $t_{15}$ , and  $P_5$ :  $t_1$ - $t_4$ - $t_6$ - $t_{11}$ - $t_{15}$ . According to poset  $(T^{exe}(\mathcal{P}), \preceq)$ , sub-path  $P_3$  needs more execution time than the other ones and consequently has to be scheduled first. According to the data of Table 2, two tasks  $t_9$  and  $t_{12}$  just can be executed by  $vm_1$ . Now, we determine the value of  $\alpha$  for the maximal node,  $t_{12}$ ;  $\alpha$  is equal to 1 in this case, which allows us to change the virtual machine of  $t_{12}$  to  $vm_2$  and update  $\alpha$  to 0. Since the value of the extra time for this task and its parent,  $t_9$ , is zero, they will not introduce any changes (see row 3 of Table 3). After removing sub-path  $P_3$  from  $W_1$ ,  $P_5$  is the only path to be scheduled. Based on the execution times of the tasks on the various virtual machines, sub-path  $P_5$  can be executed by  $vm_1$ . However, the value of extra time changes the virtual machine of tasks  $t_4$ ,  $t_6$ , and  $t_{15}$  to  $vm_2$  (see row 4 of Table 3).

### 5.3 The Role of $\alpha$

According to the explanation of the extra time, the value of  $\alpha$  is extracted from the maximal node to the minimal for each

chain examined by the GuRMiC algorithm. We exploit such time laxity to assign a slower machine if possible. Such adjustment method has some advantages and disadvantages discussed below.

Let  $t_i$  be the task that GuRMiC is processing. Leveraging  $\alpha$  can be advantageous not only by allowing the use of a slower virtual machine for  $t_i$  but also its parents in other chains. The latter is realized by delaying the start time of  $t_i$  and allowing its parents to take more time in completing their execution. However, when  $t_i$  has many child nodes, executing  $t_i$  on a slower virtual machine further constraints these children. Hence, the  $\alpha$ -time allocation process is most effective when the number of children is small. On the other hand, if  $t_i$  has no parents, we cannot utilize the advantage of allocating more time to its antecedent nodes. In such a case, we modify the  $\alpha$ -time allocation as follows (Lines 6-18 of Algorithm 2). Let  $P$ :  $t_l$ - $t_{l+1}$ -...- $t_j$  be the selected ordered chain in the algorithm. We check the nodes of  $P$  from the minimal node to the maximal. Suppose that the current node is  $t_i$ . If  $t_i$  has no unscheduled parent, we should shift back  $t_i$  because this time has a negative effect. Since there is no parent to use the advantage of the extra time, increasing the finish time of  $t_i$  by  $\alpha$  just creates a harder starting constraint for the children of  $t_i$ .

Note that the proposed extra time allocation process is not optimal because selecting two or more tasks and distributing the extra time between them may have better performance than selecting tasks from the maximal node to the minimal. Yet such a solution requires consideration of all possible permutations that is not effective and practical due to the high complexity. We have conducted some simulations to assess how GuRMiC performs relative to an optimal solution that explores all options. The results indicate that the difference is approximately 11% for small workflows.

For more clarity, we illustrate the aforementioned analysis with a simple workflow with seven tasks (see Fig. 5) and three available virtual machines  $vm_1$ ,  $vm_2$ , and  $vm_3$ . Suppose that the execution time of tasks  $t_1$  to  $t_7$  on virtual machine  $vm_1$  is equal to 2, 4, 2, 2, 3, 9, and 5, respectively. Moreover, assume that  $vm_2$  is 2 times slower than  $vm_1$ , while this ratio is 3 for  $vm_3$ . Finally, let the deadline of this workflow be 18. At first, GuRMiC schedules the critical path of the workflow, which is  $t_2$ - $t_5$ - $t_6$ . Based on the execution times and the predefined deadline, this path can be executed on  $vm_1$ . Now, we can compute the extra time for each task of this path from the maximal node to the minimal. For task  $t_6$ , the value of  $\alpha$  is 2, but we cannot change its virtual machine since it requires 9 units of extra time. Hence, we increase the start and finish time of task  $t_6$  by the value of  $\alpha$ . This process is also repeated for tasks  $t_5$  and  $t_2$ .

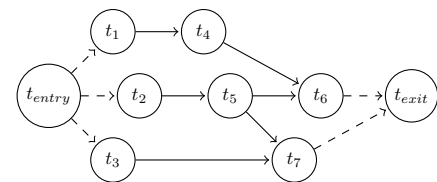


Fig. 5. A simple workflow for checking the effect of the extra time

However, the start and finish times for tasks  $t_2$  and  $t_5$



TABLE 3  
The values of start, finish, and extra times for each step of running GuRMiC on the workflow of Fig. 3.

Path	Time	Tasks															
		$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$	$t_7$	$t_8$	$t_9$	$t_{10}$	$t_{11}$	$t_{12}$	$t_{13}$	$t_{14}$	$t_{15}$	$t_{16}$
$P_1$	Start	0	0	0	2	4	7	7	7	14	14	15	15	17	8	19	22
	Finish	2	4	3	3	7	12	8	14	15	17	19	16	22	9	24	31
	Extra		0			0			0		0			0			0
$P_2$	Start	0	0	0	2	4	>7	>7	7	>14	14	15	15	17	8	19	22
	Finish	2	4	<4	<4	7	12	8	14	15	17	19	<17	22	<22	24	31
	Extra			1				11							13		
	Start	0	0	0	2	4	>7	7	7	>14	14	15	15	17	10	19	22
	Finish	2	4	3	<4	7	12	10	14	15	17	19	<17	22	13	24	31
	Extra			1				0							9		
$P_3$	Start	0	0	0	2	4	>7	7	7	>14	14	15	15	17	10	19	22
	Finish	2	4	3	<4	7	12	10	14	15	17	19	<17	22	13	24	31
	Extra								0				1				
	Start	0	0	0	2	4	>7	7	7	14	14	>15	15	17	10	19	22
	Finish	2	4	3	<4	7	12	10	14	15	17	19	17	22	13	24	31
	Extra								0				0				
$P_5$	Start	0	0	0	2	4	>7	7	7	14	14	>15	15	17	10	19	22
	Finish	2	4	3	<4	7	12	10	14	15	17	19	17	22	13	24	31
	Extra	0			1		5					2				7	
	Start	0	0	0	2	4	7	7	7	14	14	17	15	17	10	21	22
	Finish	2	4	3	4	7	17	10	14	15	17	21	17	22	13	31	31
	Extra	0			0		0					0				0	
VM		$vm_1$	$vm_1$	$vm_1$	$vm_2$	$vm_1$	$vm_2$	$vm_3$	$vm_1$	$vm_1$	$vm_1$	$vm_1$	$vm_2$	$vm_1$	$vm_3$	$vm_2$	$vm_1$

must return back to their previous values, i.e., by deducting  $\alpha$  because they have no unscheduled parent to potentially take advantage of the extra time. Not resetting the times of  $t_2$  and  $t_5$  will cause task  $t_7$  to have " $> 9$ " as the constraint for when it can start execution. Hence, in the best case, this task can be scheduled on  $vm_2$ . Meanwhile, by reducing the extra time,  $\alpha$ , the start constraint of task  $t_7$  will be " $> 7$ ". This constraint allows us to schedule  $t_7$  on  $vm_3$ , thus reducing the execution cost. Note that, for the minimal node of a chain, if we can not change the assigned virtual machine of this node to a slower one, then adding the extra time is always harmful and just creates a more considerable starting constraint for its successors.

## 5.4 Time Complexity

In the following, we assess the time complexity of our proposed GuRMiC algorithm. Let  $G = (V, E)$  be the given workflow with  $n$  tasks, and there exist  $D$  levels from sources (the children of  $t_{entry}$ ) to sinks (the parents of  $t_{exit}$ ). Let  $n_d$  be the number of vertices in  $d^{th}$  level where  $1 \leq d \leq D$ . Since  $G$  is an acyclic directed graph, we can obtain  $|E| = \sum_{d=1}^D n_d n_{d+1}$ . Therefore, we can conclude that the order of edges is given by  $O(n^2)$ . In the first step of GuRMiC, the start and finish times of all workflow nodes are calculated based on the fastest virtual machine. Hence, starting from  $t_{entry}$ , GuRMiC calculates the start and finish times of the children of  $t_{entry}$ , and repeats this process for all nodes in the workflow. This means that the time complexity of this step is  $O(n)$ . In the second step, GuRMiC forms the poset  $(T^{exe}(\mathcal{P}), \preceq)$  which includes workflow chains. Note that all elements of the chains are scheduled only once (from the longest to the shortest); therefore it is not necessary to keep a chain whose elements are duplicates in this poset. For creating  $(T^{exe}(\mathcal{P}), \preceq)$ , GuRMiC starts from the maximum

element and recursively selects the parent with the longest finish time (see Section 4.2). This recursive method causes each task to be examined only once and chains with at least one new element to be created. Thus, the time complexity of creating the poset is  $O(n)$ .

In the next step, GuRMiC examines the virtual machines to find the slowest one that does not violate the deadline and constraints. Since the processing speed of a virtual machine is determined by its type, as mentioned in Section 2.2, it is sufficient to consider  $m$  virtual machine options for each task. As we explained, the start and finish times of each task are calculated only once; hence the time complexity of finding an appropriate virtual machine for all workflow tasks is  $O(mn) \approx O(n^2)$ . After that, GuRMiC checks the impact of the extra time on each task of the selected chain. Without loss of generality, we consider the overall behavior of this process instead of factoring in all details. Overall, GuRMiC computes the value of the extra time at most twice (once before changing the virtual machine and once after that) and also shifts the start and finish times of each task at most twice (once for providing more time for parents and once for returning back a task that does not have an unscheduled parent to the previous position). On the other hand, this process is performed only once for each task in the workflow; hence its time complexity is  $O(n)$ . In the final step of GuRMiC, the start and finish constraints are assigned to all successors and predecessors connected to the nodes of the selected path. We know that each node is scheduled only once, and in the worst case, each node can be connected to  $n - 1$  nodes; thus the time complexity of this step is  $O(n^2)$ . Consequently, the overall time complexity of GuRMiC is  $O(n^2)$ . Although such time complexity resembles that of the competing methods, the simulation results presented earlier, have confirmed the advantages of GuRMiC.

## 6 PERFORMANCE EVALUATION

### 6.1 Simulation Settings and Baseline Approaches

**Simulation Environment:** We use the WorkflowSim [9] to implement the proposed method and simulate the cloud environment. WorkflowSim is an open-source platform developed by the University of Southern California's Pegasus WMS group. It supports widely accepted features of workflows and also heterogeneous system overheads and failures. In our experiments, the characteristics of the virtual machines are based on the Amazon EC2 instances [10], and listed in Table 4. In this table, *ECU* is the equivalent CPU power of a 1.0-1.2 GHz 2007 Opteron or Xeon processor. Moreover, similar to Amazon EC2, the bandwidth for communication among VMs is set to 20 Mbps.

TABLE 4  
The Amazon EC2 instance specifications

Type	Core Speed (ECU)	Processing Cores	RAM (GB)	Storage (GB)	Cost per hour (\$)
m1.small	1	1	1.7	160	0.6
m1.large	4	2	7.5	850	0.24
m1.xlarge	8	4	15	1690	0.48
c1.medium	5	2	1.7	350	0.30
c1.xlarge	20	8	7	1690	1.20

**Workflows:** In our simulations, we use three widely-used sets of scientific workflows to evaluate the performance, namely, Montage, LIGO, and Epigenomics. The Montage workflows are I/O intensive, but they do not require much CPU processing capacity. These workflows are utilized to generate custom mosaics of the sky according to a set of input images. The LIGO workflows include CPU-intensive tasks with high memory requirements in order to detect gravitational waves. In the bioinformatics domain, the Epigenomics workflows are used to automate the execution of various genome-sequencing operations. They constitute examples of CPU-intensive applications. Fig. 6 shows the structures of these workflows (for a complete description, see [11]). Moreover, we evaluate the performance based on different workflow sizes; we chose three sizes, small (up to 100 tasks), medium (100 to 500 tasks), and large (500 to 1000 tasks), in our experiments.

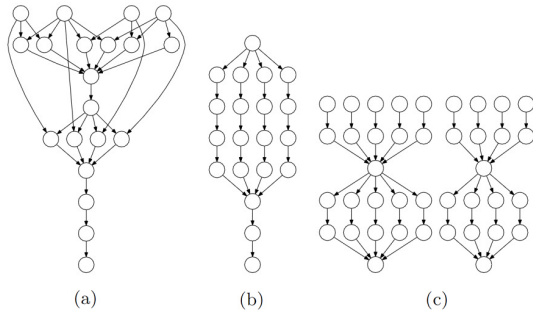


Fig. 6. Workflow structures: (a) Montage, (b) Epigenomics, and (c) LIGO.

**Baseline Approaches:** As will be highlighted in Section 7, existing work can be categorized based on the optimization objective and constraints. The variation is mainly motivated by application requirements. GuRMiC opts to minimize the execution cost of a feasible scheduling while satisfying a

predefined deadline. To evaluate the effectiveness of GuRMiC, we compare its performance to published schemes that tackle the same problem, i.e., have the same objective and constraint. Therefore, deadline-constrained-based cost optimization methods are considered for comparison. To this end, GuRMiC is compared with IC-PCP, IC-PCPD2 [7], CTDC [12], and LBWS [13] to demonstrate the influence of considering poset  $(T^{exe}(\mathcal{P}), \preceq)$  in improvement of the results. These selected schemes reflect the best known algorithms in the literature for the underlying optimization problem. Note that in [13], the authors proposed two approaches LBWS and SAWS; they both utilize the concept of Comparative Advantage (CA) as a metric. The value of CA indicates the number of times that a method creates a schedule with a lower cost than the other. Their results show that LBWS has better CA than SAWS; hence we compare GuRMiC with LBWS. The reported results reflect the average over 30 simulation runs.

### 6.2 Results and Analysis

In this section, we compare our approach with IC-PCP, IC-PCPD2, CTDC, and LBWS to demonstrate the superiority of the proposed approach in terms of the execution cost. Since the workflows, which are used to evaluate the performance of the approaches, have different attributes such as type and task size, it is important that the total cost of each workflow execution is normalized. Normalized cost (NC) of a workflow  $W$  is defined as follows:

$$NC(W, app) = \frac{C_{app}(W)}{C_c(W)} \quad (7)$$

where  $C_{app}(W)$  is the total execution cost of  $W$  by approach  $app$  and  $C_c(W)$  indicates the cost of scheduling all tasks of  $W$  on the cheapest virtual machine.

Figs. 7 shows the normalized cost for the compared approaches when the deadline factor equals 1.5 and 2. As expected, increasing the deadline factor causes a decrease in the normalized cost because the workflow tasks have more extra time and can be scheduled on a cheaper virtual machine. The results imply that GuRMiC outperforms all competing approaches for LIGO and Epigenomics workflows. According to the results, the performance of CTDC is better than IC-PCP, IC-PCPD2, and LBWS for these two workflows. In particular, considering a different process for distributing the total extra time is the main reason for the good performance of GuRMiC and CTDC. CTDC uses two following methods to distribute the total extra time among the tasks in a path: (1) distributing time equally among all the tasks, and (2) distributing time in proportion to the length (i.e., the number of instructions) of a task. CTDC examines these two methods and chooses the one which reduces the total execution cost the most. Although this process can boost the performance, it restrains the potential improvement. Specifically, the first method, can give each task too little extra time to allow switching to a better virtual machine, especially when the deadline is tight. A similar scenario can be pointed out for the second method. GuRMiC avoids this shortcoming and considers the total extra time from the maximal node to the minimal. Although this consideration is not optimal, the results show that it improves the normalized costs more than the other methods.

Moreover, the results show that CTDC performs the worst for Montage. When examining the structure of Montage in Fig. 6, one can note that many tasks with an approximately equal execution time are located at the second level of the corresponding DAG. CTDC considers the extra time for tasks based on the difference between the earliest and latest start times of the task. Since such difference for the second level tasks of Montage is small, CTDC considers a short extra time for each of these tasks. As a result, CTDC cannot schedule these tasks on low-cost virtual machines. The performance of LBWS contrasts with CTDC for Montage. Moreover, the normalized cost of LBWS is better than our approach. The reason is that LBWS allocates larger sub-deadlines for tasks in the second level of this workflow; thus, they can be serialized and scheduled on the same machine, which reduces the total execution cost. Meanwhile, due to the structure of poset  $(T^{exe}(\mathcal{P}), \preceq)$ , using such a distribution of extra time is not possible in GuRMiC. In fact, GuRMiC computes the extra time for the tasks of a workflow path based on the poset to find the slowest possible virtual machines for them; thus, GuRMiC needs to launch more virtual machines for tasks in the second level of Montage where the time intervals of these machines are not fully utilized. Hence, LBWS is a better solution for workflows with a parallel structure like Montage (however, Section 6.3 shows that the performance of LBWS and GuRMiC is not significantly different), while GuRMiC is more suited for workflows that are composed of multiple pipelines (such as Epigenomics). On the other hand, the normalized cost of LBWS is the highest for the LIGO workflow. This is due to the process of distributing the total extra time among tasks based on the level, while in the other approaches, it is distributed to tasks belonging to the same path.

### 6.3 Statistical Significance

We have validated the statistical significance of the experimental results by applying the one-way analysis of variance (ANOVA test) [14]. Such a test compares the means of three or more groups and determines whether these means are significantly different from each other. The null and alternative hypotheses of ANOVA are:

- $H_0 : \forall \phi, \psi \in \mathcal{A} : \mu_\phi = \mu_\psi$
- $H_1 : \exists \phi, \psi \in \mathcal{A} : \mu_\phi \neq \mu_\psi \wedge \phi \neq \psi$

where  $\mathcal{A}$  is the set of compared approaches (i.e., IC-PCP, IC-PCPD2, CTDC, LBWS, and GuRMiC) and  $\mu_{app}$  indicates the sample mean for approach  $app$ . More precisely, the null hypothesis,  $H_0$ , of ANOVA implies no difference among the means, and the alternate hypothesis,  $H_1$ , indicates that the means are different from one another. Table 5 demonstrates the statistical results of the compared approaches for all large-size workflows where SV, SS, df, MS, and F represent the source of variation, the sum of squares, the degrees of freedom, the mean square, and the F-test, respectively. Moreover, in column SV, notions BG, WG, and T denote between-groups, within-groups, and total variations. Let the significance level be 0.05. Since the probability value (p-value) of all cases is less than the significance level, we reject the null hypothesis and accept  $H_1$ , which states that the mean of at least one approach is different from the others.

TABLE 5  
The results of one-way ANOVA test

Workflow	SV	SS	df	MS	F	p-value
Montage	BG	48720.74	4	12180.185	13.751986	1.56E-09
	WG	128427	145	885.70368		
	T	177147.8	149			
LIGO	BG	76.05965	4	19.01491	138.1982	1.98E-48
	WG	19.95079	145	0.137592		
	T	96.01044	149			
Epigenomics	BG	31.83554	4	7.958884	599.8826	4.30E-89
	WG	1.923773	145	0.013267		
	T	33.75931	149			

Although the results support hypothesis  $H_1$ , all compared approaches are not necessarily different from each other. Hence, we need to perform other types of tests, known as post-hoc tests, to determine which pairwise comparisons are significant. In general, there exist  $C(\kappa, 2) = \kappa(\kappa + 1)/2$  pairwise tests where  $\kappa$  is the number of samples, so for each workflow, we have  $C(5, 2) = 15$  pairwise tests. Since our goal is to show the difference between GuRMiC and other competing methods, we consider only 4 comparisons that include our approach (see Table 6). To this end, the Tukey Honestly Significant Difference (Tukey HSD) test is used as a post-hoc test. The relevant statistic is

$$q = \frac{|\mu_\phi - \mu_\psi|}{\sqrt{MS_W/m}} \quad (8)$$

where  $\sqrt{MS_W/m}$  indicates the standard error, in which  $MS_W$  and  $m$  are the mean square within groups and the size of each of the group samples, respectively. The statistic  $q$  has a Studentized Range Distribution, and the critical values for this distribution (called  $q_c$ ) are obtained from the Studentized Range  $q$  Table<sup>1</sup> based on the significant levels, the number of groups ( $\kappa$ ), and the degrees of freedom within groups ( $df_W$ ). Note that if  $q > q_c$ , then the two means are significantly different. Table 6 shows the results of the Tukey HSD test for all large-size workflows, where the "S/NS" column represents the significance of the difference between GuRMiC and the compared approaches. Using the Studentized Range  $q$  Table with significant level of 0.05,  $\kappa = 5$  and  $df_W = 145$ , we get  $q_c = 3.902$ . Hence, if GuRMiC and the compared approach are significantly different (i.e.,  $q > 3.902$ ), the value of the "S/NS" column is S; otherwise, the value is NS. The results show that in most of the cases, the proposed approach is significantly different from the compared approaches according to Tukey HSD. However, in Montage, the normalized cost of LBWS is better than our approach, but Tukey HSD demonstrates that the performance of these two approaches is not significantly different. There is a similar description for the better normalized cost of GuRMiC compared to CTDC in LIGO (however, the value of  $q$  in CTDC is very close to the critical value). According to the results, it can be concluded that there is sufficient evidence that the proposed approach significantly outperforms the other methods.

1. See the details of the Studentized Range  $q$  Table at [www.real-statistics.com/statistics-tables/studentized-range-q-table/](http://www.real-statistics.com/statistics-tables/studentized-range-q-table/)



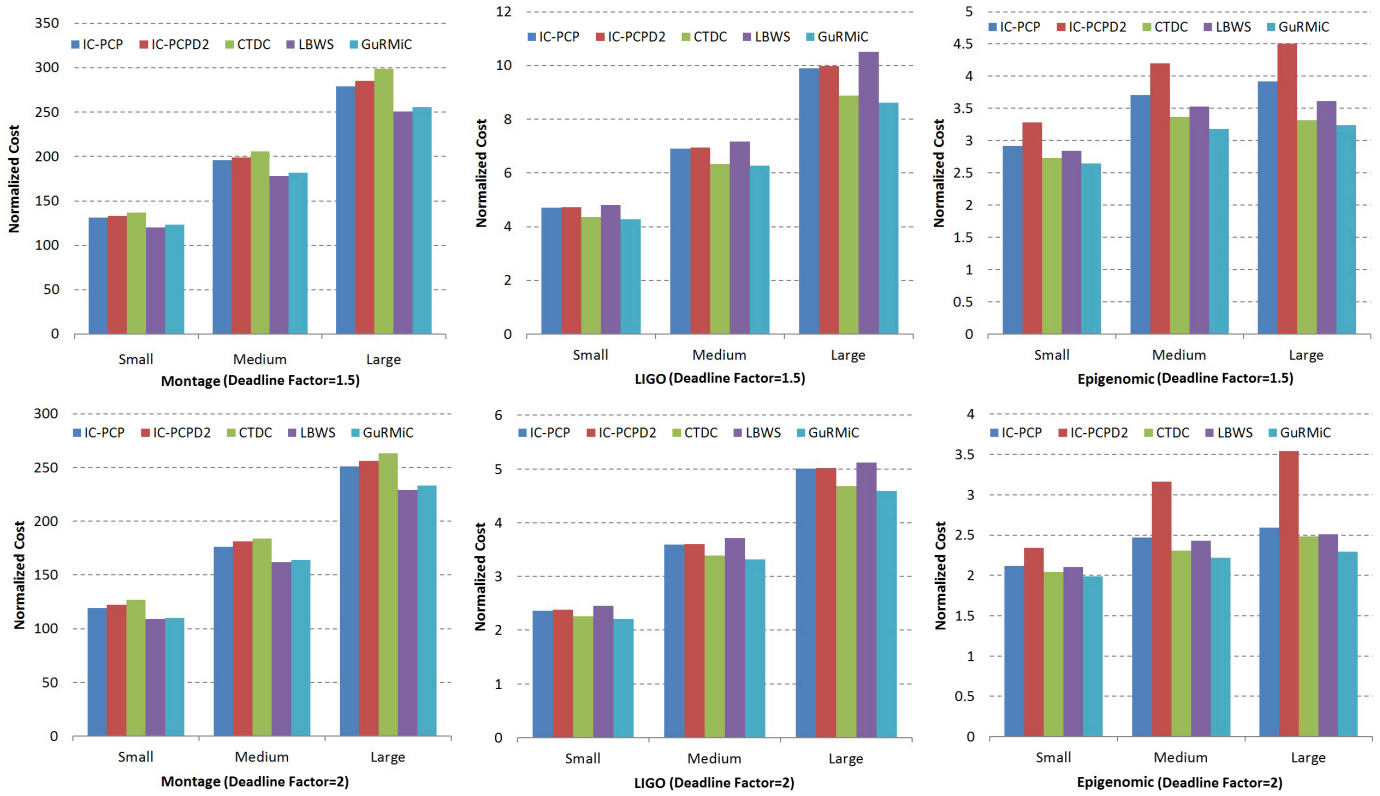


Fig. 7. The normalized cost of GuRMiC, IC-PCP, IC-PCPD2, CTDC, and LBWS with the deadline factor equals to 1.5 and 2

TABLE 6

The Tukey HSD test results for GuRMiC compared to other approaches

Approach	Metric	Workflow		
		Montage	LIGO	Epigenomics
IC-PCP	q	4.232960716	18.91040029	32.82664681
	S/NS	S	S	S
IC-PCPD2	q	5.337395379	20.08676303	60.39088572
	S/NS	S	S	S
CTDC	q	7.913918816	3.701358932	4.247967815
	S/NS	S	NS	S
LBWS	q	0.920086157	27.77003603	18.06971384
	S/NS	NS	S	S

## 7 RELATED WORK

Given the focus of GuRMiC, we review approaches that consider the trade-off between the execution latency and cost; hence, schemes that predominantly consider other metrics, e.g., energy [15], are not covered. We categorize the related work based on the main optimization objective. Table 7 provides a comparative summary of the cited references.

**Deadline-centric Scheduling:** The notion of a partial critical path (PCP) is introduced in [16], where tasks on the critical path are assigned deadlines so that they can be independently scheduled. IC-PCP and IC-PCPD2 use such a notion [7]. IC-PCP separately schedules each critical task, while IC-PCPD2 takes advantage of the deadline laxity of a task to reduce the cost of executing the next task on the critical path. They cannot guarantee the fulfillment of hard deadlines [13]. IC-PCP is improved in [17] through An iterative algorithm to increase the probability of, rather

than guarantee, deadline satisfaction. Instead of using PCP, multiple complete critical paths are formed in CPI [18]. CPI maps the critical path optimization to a multi-stage decision process and solves it by dynamic programming based Pareto method, which is inefficient.

The scheduling strategy of Han et al. is to decouple the precedence and deadline constraints from the VM assignment [13]. Two scheduling algorithms are proposed, namely, LBWS and SAWS. In LBWS, tasks are assigned deadlines based on the overall workflow makespan requirement. Meanwhile, SAWS allocates tasks one level at a time using the VMs with minimum cost. The topological order of the workflow tasks is guaranteed by level ordering, and earlier execution of the critical path is guaranteed by the associated sub-deadline. CETSS also pursues level-based task assignment [19], yet based on multiple parameters rather than only cost. Cai et al. [20] consider the workflow scheduling for non-shareable and shareable services with interval based pricing models. Hence, they divide the problem into (i) task mode mapping, and (ii) task tabling on renting service instances. Two heuristics, namely, CPIS and LHCM, are proposed for these sub-problems, respectively. Meanwhile, variation of the execution time requirement of the individual tasks is considered in [21]. A time-series model is employed to overcome such variability. Unlike the approaches discussed above, Li et al. assume discretized time frames where a task start time has to be aligned with time slots of fixed lengths [22]. The scheduling problem is modeled as a mixed integer program.

**Cost-constrained Workload Scheduling:** PCP-B<sup>2</sup> opts to minimize the makespan while satisfying the budget con-



straint [23]. However, PCP-B<sup>2</sup> is only efficient in scheduling pipeline structured workflows. CB-DT [24] and AILS [25] tackle the same problem using a backtracking method and iterated local search, respectively. Arabnejad et al. proposed two algorithms, PDC and DCCP [26]. The PDC algorithm separates a workflow into logical levels and then subdivides the overall user-defined deadline over these levels to maximize parallelism. DCCP leverages the PCP concept, discussed earlier, to schedule all critical tasks on the same virtual machine to reduce the communication costs.

Poola et al. [27] and Matani et al. [28] have studied a variant of the scheduling problem where the task execution time is subject to some uncertainty. Meanwhile, a more complicated problem is considered in [29], by assuming uncertainty about the available cloud resources. However, the focus is only on finding Pareto optimal solutions. On the other hand, some studies have considered both budget and completion time constraints. While BDAS [30] employs a search heuristic, the solutions of [29] and [31] are Genetic Algorithm based. Finally, some work has considered other optimization metrics in addition to coping with budget and deadline constraints. For example, in [32] two heuristic algorithms are proposed to minimize energy consumption without exceeding completion time and budget bounds.

**Multi-Objective and Unconstrained Optimization:** Some of the published studies considered more than one metric in optimizing the workflow schedule with and without factoring in time constraints. Bugingo et al. opt to minimize cost and energy in a cloud environment [33]. A similar variant of the scheduling problem is tackled in [34], by optimizing energy consumption, execution cost, and resource utilization in cloud data centers. In [35], Li et al. present a deadline constrained workflow scheduling algorithm with the aim of minimizing the execution cost and energy in the cloud. EViMA strives to do the same yet while minimizing the workflow makespan as well [36].

Ghafouri et al. proposed two heuristics, MTDC and CTDC, to decrease the cost of executing a workflow [12]. Unlike most published schemes, both algorithms handle a workflow represented using a general task graph, where a task may have multiple parents. CEFA [37] pursues a meta-heuristic rather than a search-heuristic based solution. However, the convergence of such meta-heuristic is unbounded and consequently meeting hard deadlines is guaranteed. Overall, existing work whether it deals with cost and execution time as objectives or constraints care for the critical path at the expense of the optimization of non-critical tasks. Such a shortcoming leads to either inability to achieve optimal performance or possibility of missing some hard deadlines, as we explained earlier.

## 8 CONCLUSIONS

This paper has presented a novel workflow scheduling algorithm based on a totally order set  $(T^{exe}(\mathcal{P}), \preceq)$  and an algebraic structure on workflow tasks with two operators  $\wedge$  and  $\vee$ . The algorithm assumes a graph-modeled workflow and pursues a divide-and-conquer strategy. First, the tasks on the critical path are assigned to the least cost virtual machines while meeting the workflow deadline. The

TABLE 7  
Summary of workflow scheduling schemes; in all time complexities,  $n$  denotes the number of workflow tasks (for more details about other parameters, see the corresponding reference)

Algorithm	Objective(s)	Constraint		Time Complexity
		Deadline	Budget	
GuRMIC	Cost	×		$O(n^2)$
PCP [16]	Cost	×		$O(n^3m)$
IC-PCP & IC-PCPD2 [7]	Cost	×		$O(n^2)$
EIPR [17]	Cost	×		$N/A$
CPI [18]	Cost	×		$O(n^3d^2m)$
LBWS & SAWS [13]	Cost	×		$O(n^2k)$
CETSS [19]	Cost	×		$O(n^3)$
CIPS & LHCM [20]	Cost	×		$O(n^3D^2W)$
CPDE [21]	Cost	×		$O((m+m_1)n^2)$
ILAH [22]	Cost	×		$O(n^2 \ln n)$
PCP-B <sup>2</sup> [23]	Makespan		×	$O(n^2 \log m)$
CB-DT [24]	Makespan		×	$O(n^2)$
AILS [25]	Makespan		×	$N/A$
DCCP [26]	Cost	×		$O(n^2p)$
RCT/RTC [27]	Makespan and Cost	×	×	$O(n^2)$
MLCP [28]	Makespan		×	$N/A$
BDAS [30]	Makespan and Cost	×	×	$N/A$
DCHG-TS [31]	Makespan and Cost	×	×	Metaheuristic
Calzarossa et al. [29]	Makespan and Cost	×	×	Metaheuristic
BDCE & BDD [32]	Cost and Energy	×	×	$O(n^2)$
Bugingo et al. [33]	Cost and Energy	×		$N/A$
ECWS [34]	Cost, Energy, and Resource Utilization	×		$O(n^2v)$
CEAS [35]	Cost and Energy	×		$O(Kn^2(n+e))$
EViMA [36]	Makespan, Cost, and Energy	×		$N/A$
CTDC [12]	Cost	×		$O(n^2)$
MTDC [12]	Makespan and Cost			$O(n^2)$
CEFA [37]	Makespan and Cost	×		Metaheuristic

handled path is then deleted and the resulting blocks (sub-workflows) are considered. Mapping a workflow to an algebraic structure with a totally order set  $(T^{exe}(\mathcal{P}), \preceq)$  helps to preserve the logical relation between the main workflow and sub-workflows in each step. The experimental results with statistical tests show that the proposed algorithm reduces the cost and outperforms the state-of-the-art methods. Although the proposed approach targets one of the most challenging objectives from the user's point of view, it does not consider the interest of service providers, where they opt to minimize energy consumption and maximize resource utilization. In fact, optimizing these objectives can help cloud providers to reduce their operation costs. Hence, in the future we plan to extend the proposed approach to consider complex cost models and other objectives such as energy. We also intend to define an additional structure for workflows to distribute the overall extra time between workflow levels, which allows us to serialize parallel tasks in a workflow when applicable.

## ACKNOWLEDGMENTS

The authors would like to express special thanks to Mahallat Institute of Higher Education for supporting this study.

## REFERENCES

- [1] G. Juve, A. L. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.
- [2] S. Doostali, S. M. Babamir, and M. Eini, "Cp-pgwo: multi-objective workflow scheduling for cloud computing using critical path," *Cluster Computing*, pp. 1–21, 2021.
- [3] M. J. Nadjafi-Arani and S. Doostali, "A workflow scheduling algorithm based on lattice theory," in *2021 52nd Annual Iranian Mathematics Conference (AIMC)*, 2021, pp. 71–73.
- [4] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [5] "Amazon elastic compute cloud (amazon ec2)," <https://aws.amazon.com/ec2/>, online; Accessed: 2022-03-15.

- [6] M. Masdari, S. ValiKardan, Z. Shahi, and S. I. Azar, "Towards workflow scheduling in cloud computing: A comprehensive analysis," *Journal of Network and Comp. App.*, vol. 66, pp. 64–82, 2016.
- [7] 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, 2013.
- [8] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order, Second Edition*. Cambridge University Press, 2002.
- [9] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *2012 IEEE 8th International Conference on E-Science*, 2012, pp. 1–8.
- [10] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Cloud Computing - First International Conference, CloudComp 2009, Munich, Germany, October 19-21, 2009 Revised Selected Papers*, 2009, pp. 115–131.
- [11] G. Juve, A. L. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Comp. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.
- [12] R. Ghafouri, A. Movaghar, and M. Mohsenzadeh, "Time-cost efficient scheduling algorithms for executing workflow in infrastructure as a service clouds," *Wirel. Pers. Commun.*, vol. 103, no. 3, pp. 2035–2070, 2018.
- [13] P. Han, C. Du, J. Chen, and X. Du, "Minimizing monetary costs for deadline constrained workflows in cloud environments," *IEEE Access*, vol. 8, pp. 25 060–25 074, 2020.
- [14] R. Gunst, "Regression and ANOVA: an integrated approach using SAS software," *Technometrics*, vol. 45, no. 2, pp. 170–171, 2003.
- [15] M. Hussain, L.-F. Wei, A. Rehman, F. Abbas, A. Hussain, and M. Ali, "Deadline-constrained energy-aware workflow scheduling in geographically distributed cloud data centers," *Future Generation Computer Systems*, vol. 132, pp. 211–222, 2022.
- [16] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Trans. Parallel Distributed Syst.*, vol. 23, no. 8, pp. 1400–1414, 2012.
- [17] R. N. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Trans. Parallel Distributed Syst.*, vol. 25, no. 7, pp. 1787–1796, 2014.
- [18] Z. Cai, X. Li, and J. N. D. Gupta, "Critical path-based iterative heuristic for workflow scheduling in utility and cloud computing," in *Proc. 11<sup>th</sup> Serv. Oriented Comp. Conf., Berlin, Germany, 2013*.
- [19] X. Tang, W. Cao, H. Tang, T. Deng, J. Mei, Y. Liu, C. Shi, M. Xia, and Z. Zeng, "Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds," *IEEE Trans. Parallel Dist. Syst.*, vol. 33, no. 9, pp. 2079–2092, 2022.
- [20] Z. Cai, X. Li, and J. N. D. Gupta, "Heuristics for provisioning services to workflows in xaas clouds," *IEEE Trans. Serv. Comput.*, vol. 9, no. 2, pp. 250–263, 2016.
- [21] Y. Pan, S. Wang, L. Wu, Y. Xia, W. Zheng, S. Pang, Z. Zeng, P. Chen, and Y. Li, "A novel approach to scheduling workflows upon cloud resources with fluctuating performance," *Mob. Networks Appl.*, vol. 25, no. 2, pp. 690–700, 2020.
- [22] X. Li, L. Qian, and R. Ruiz, "Cloud workflow scheduling with deadlines and time slot availability," *IEEE Trans. Serv. Comput.*, vol. 11, no. 2, pp. 329–340, 2018.
- [23] F. Wu, Q. Wu, Y. Tan, R. Li, and W. Wang, "Pcp-b<sup>2</sup>: Partial critical path budget balanced scheduling algorithms for scientific workflow applications," *Future Gener. Comput. Syst.*, vol. 60, pp. 22–34, 2016.
- [24] R. Ghafouri, A. Movaghar, and M. Mohsenzadeh, "A budget constrained scheduling algorithm for executing workflow application in infrastructure as a service clouds," *Peer Peer Netw. Appl.*, vol. 12, no. 1, pp. 241–268, 2019.
- [25] S. Qin, D. Pi, and Z. Shao, "AILS: A budget-constrained adaptive iterated local search for workflow scheduling in cloud environment," *Expert Syst. Appl.*, vol. 198, p. 116824, 2022.
- [26] V. Arabnejad, K. Bubendorfer, and B. Ng, "Scheduling deadline constrained scientific workflows on dynamically provisioned cloud resources," *Fut. Gener. Comp. Sys.*, vol. 75, pp. 348–364, 2017.
- [27] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, "Robust scheduling of scientific workflows with deadline and budget constraints in clouds," in *28<sup>th</sup> IEEE Int'l Conf. on Advanced Info. Net. and App. (AINA 2014), Victoria, BC, Canada, May 2014*.
- [28] A. Matani and A. Darvishy, "A novel critical-path based scheduling algorithm for stochastic workflow in distributed computing systems," in *International Congress on High-Performance Computing and Big Data Analysis*, 2019, pp. 476–489.
- [29] M. C. Calzarossa, M. L. D. Vedova, L. Massari, G. Nebbione, and D. Tessa, "Multi-objective optimization of deadline and budget-aware workflow scheduling in uncertain clouds," *IEEE Access*, vol. 9, pp. 89 891–89 905, 2021.
- [30] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget and deadline aware e-science workflow scheduling in clouds," *IEEE Trans. Parallel Distributed Syst.*, vol. 30, no. 1, pp. 29–44, 2019.
- [31] A. Iranmanesh and H. R. Naji, "DCHG-TS: a deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing," *Clust. Comput.*, vol. 24, no. 2, pp. 667–681, 2021.
- [32] A. Taghinezhad-Niar, S. Pashazadeh, and J. Taheri, "Energy-efficient workflow scheduling with budget-deadline constraints for cloud," *Computing*, vol. 104, no. 3, pp. 601–625, 2022.
- [33] E. Bugingo, et al., "Deadline-constrained cost-energy aware workflow scheduling in cloud," *Concurr. Comput. Pract. Exp.*, vol. 34, no. 6, 2022.
- [34] R. Medara, R. S. Singh, and M. Sompalli, "Energy and cost aware workflow scheduling in clouds with deadline constraint," *Concurr. Comput. Pract. Exp.*, vol. 34, no. 13, 2022.
- [35] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, and B. Luo, "Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds," *IEEE Trans. Serv. Comput.*, vol. 11, no. 4, pp. 713–726, 2018.
- [36] J. K. Konjaang, J. Murphy, and L. Murphy, "Energy-efficient virtual-machine mapping algorithm (evima) for workflow tasks with deadlines in a cloud environment," *Journal of Network and Computer Applications*, vol. 203, p. 103400, 2022.
- [37] K. K. Chakravarthi, L. Shyamala, and V. Vaidehi, "Cost-effective workflow scheduling approach on cloud under deadline constraint using firefly algorithm," *Appl. Intell.*, vol. 51, no. 3, pp. 1629–1644, 2021.



**Mohammad Javad Nadjafi-Arani** received the Ph.D. degree in mathematics from University of Kashan, Iran, in 2012. He is currently an Assistant Professor at Mahallat Institute of Higher Education, Iran. His current research interests include distance graph theory and applications of graph modeling and probability in computer science, chemistry, biological systems, and economy. Dr. Nadjafi-Arani is a life member of the American Math. Society and European Society of Mathematical Chemistry.



**Saeed Doostali** received B.Sc. and M.Sc. degrees from Arak University, Iran, in 2009 and 2011, respectively, and a Ph.D. from the University of Kashan, Iran, in 2021, all degrees are in computer engineering. He chaired the computer department at Mahallat Institute of Higher Education from 2014 to 2019. His research interest includes distributed systems, and network optimization, especially in the cloud, IoT, application of soft computing, software testing, and formal verification in different fields of science.



**Mohamed F. Younis** is currently a professor in the CSEE at UMC. His technical interest includes network architectures and protocols, wireless networks, embedded systems, and secure communication. He has published over 330 technical papers in refereed conferences, books and journals, and has seven granted and three pending patents. He serves/served on the editorial board of multiple journals and the committees of numerous conferences. Dr. Younis is a Fellow of the IEEE and the IEEE comm. society.