Cluster Computing (2021) 24:343–359
https://doi.org/10.1007/s10586-020-03109-y

Task replication to improve the reliability of running workflows on the cloud

Samaneh Sadat Mousavi Nik[1] · Mahmoud Naghibzadeh[1] · Yasser Sedaghat[1]

Received: 24 August 2019 / Revised: 17 January 2020 / Accepted: 10 April 2020 / Published online: 27 April 2020

Abstract

Clouds are heterogeneous service-oriented systems which are increasingly considered as platforms of choice for scientific workflow applications. Because resource and communication failures are inevitable in large complex distributed systems, insuring the reliability of heterogeneous service-oriented systems poses a major challenge. As it affects the quality of user service requirements, reliability has become an important criterion in workflow scheduling. Replication-based fault-tolerance is one approach for satisfying the requirements set to safeguard the reliability of an application. In order to minimize the workflow execution cost while respecting the user-defined deadline and reliability, the present paper proposes Improving CbCP with Replication (ICR) which includes three algorithms: the Scheduling, the Fix Up, and the Task Replication. The Scheduling employs the CbCP algorithm, where CbCP stands for Clustering based on Critical Parent and it is a previously developed algorithm by the same authors, to generate a schedule map of the workflow. The Fix Up algorithm checks the possibility of starting each task earlier in the leased resource without imposing any extra cost. The Task Replication algorithm utilizes the rest of the idle time slots in leased resources to replicate tasks. Experimental results from real and randomly generated applications at different scales demonstrate that the proposed heuristic, for the majority of studied scenarios, increases the execution reliability of workflows while reducing the workflows execution costs.

Keywords Cloud computing · Reliability · QoS-based scheduling · Task replication

1 introduction

Scientific workflows are often designed as directed acyclic graphs (DAGs) which nodes act for tasks and directed edges signify dependencies between tasks. As a single workflow can hold hundreds or thousands of tasks [1], this kind of workflow can benefit from large-scale infrastructures, such as the public Cloud. Cloud computing has provided an infrastructure for the sharing of large scale and heterogeneous resources, for instance, Virtual Machines and data resources. In large-scale heterogeneous systems, resource management is a critical issue due to the various configurations and capacities of hardware and software. In order to carry out scientific applications, which are usually modeled by workflows on the Cloud, the efficient scheduling of algorithms is necessary to meet user or system requirements [2]. Since users are charged for a workflow's execution on a Cloud, cost is a Quality of Service (QoS) parameter. In this parameter, the number of time intervals consumed by users determines how a majority of today's commercial Clouds set prices. In addition, the processing capacity of VMs in heterogeneous service-oriented systems has been advanced to supply powerful Cloud-based services, while the failures of VMs and communication faults [3] affect the reliability of systems and the quality of service for users [4]. As a result, reliability has become another critical challenge in workflow scheduling [4–7]. Moreover, because the resources available to an application can provide dynamic scaling in response to demands, Cloud infrastructures are suitable platforms for the execution of deadline-constrained

✉ Mahmoud Naghibzadeh
naghibzadeh@um.ac.ir

Samaneh Sadat Mousavi Nik
sa.mousavinik@mail.um.ac.ir

Yasser Sedaghat
y_sedaghat@um.ac.ir

[1] Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

workflow applications. With all this variety, application developers hosting their system in the Cloud are challenged to make the best possibility in terms of price, reliability, and deadline.

Workflow scheduling algorithms plays a critical role in meeting QoS requirements. Scheduling is defined as the mapping of tasks to processors so that specific requirements are satisfied, while still meeting the requirements of task precedence. As optimal task scheduling is NP-Complete [8], several heuristic methods have been introduced for homogeneous [9] and heterogeneous distributed systems [10, 11]. Accordingly, heuristics can be utilized to obtain sub-optimal schedules. General task scheduling algorithms are categorized into various classes, such as list scheduling algorithms, cluster algorithms, and duplication-based algorithms. For example, HEFT is a list-scheduling algorithm [9] which selects tasks according to the descending order of their upward ranks and then dispatches them to different VMs with the objective of reducing the execution length.

In the Cloud, other factors also hold great importance, such as workflow execution reliability and execution costs. Reliability is determined as the probability of a schedule successfully completing its execution [4–7]. If an application can satisfy its identified reliability requirement, then it is considered as reliable. A number of algorithms have been proposed to improve reliability by redundancy in space and time [12, 13]. One of the popular methods for producing fault-tolerance is redundancy in space. For instance, a replication scheme is applied in [3] to maximize execution reliability. In the active replication scheme, each task is simultaneously replicated on several VMs. A task will succeed if at least one replication does not fail [14–16]. However, more replicas signify increased resource usage and greater costs. Whence, the reliability problem of service-oriented systems is predominantly satisfying an application's reliability demand whilst still reducing the use of resources as much as possible.

Another technique that can mitigate failures is the backup/restart scheme, in which a task is rescheduled on a backup VM in order to proceed when a VM fails [17, 18]. An improved version of the backup/restart method is the checkpoint/restart scheme. In this scheme, when a failure occurs, the task can be restarted from the latest checkpoint instead of from the very beginning. Both the backup/restart and checkpoint/restart schemes are based on redundancy in time, which can result in missing the deadline [5].

In [19], the authors proposed Clustering based on Critical Parent (CbCP) scheduler, in which the probability of failure for a cluster on a resource is known. Then, different clusters are assigned onto corresponding resources by considering each cluster's sub-reliability requirement.

The main procedures of CbCP are as follows:

(1) Dividing the workflow into a number of Clusters based on a Critical Parent (CbCP) then the reliability requirement of the application is transferred to the sub-reliability requirements of the clusters. In this way, as long as the sub-reliability requirement of each cluster can be satisfied, so can the application's reliability requirement. A workflow is made up of a quantity of tasks, with each task's sub-deadline related to the workflow's general deadline D. As a result, exit tasks have sub-deadlines equal to D, while a traversal of the workflow graph in reverse topological order determines the remaining tasks' sub-deadlines, in such a way that a heuristic assignment can be utilized in the following.

(2) CbCP iteratively assigns each cluster to processors with a minimum execution cost up to the sub-reliability demand and sub-deadline of the cluster is satisfied.

Finding the cheapest schedule that can complete each task of the workflow prior to its Latest Finish Time is the goal of the CbCP's approach. Therefore, CbCP attempts to postpone the start time of tasks as much as possible, which consequently may cause an increase in the workflow execution cost. In contrast, the currently proposed algorithm checks the possibility of starting each task earlier in the leased resource if no extra cost is imposed, by shifting each task toward the beginning of the time interval. Moreover, CbCP is not considering idle time slots to increase the overall reliability, while the present work's algorithm explores the possibility of task replication to increase workflow execution reliability. These replicas also increase resource utilization at no extra cost.

Based on previous workflow scheduling research in the context of the cloud, scheduling scientific workflows on the cloud extremely reduces costs and makespan. However, like any other distributed system, cloud computing is also prone to resource failures, such as with VMs and relevant network resources. As a result, considering the probability of failure of those resources is imperative.

To address this problem, the current study proposes a new algorithm called ICR (Improving CbCP with Replication). ICR includes three algorithms: the Scheduling, the Fix Up, and the Task Replication. The Scheduling employs the CbCP algorithm to generate a schedule map of the workflow. The Fix Up algorithm attempts to explore the possibility of starting each task earlier in the leased resource, without imposing any extra cost. The Task Replication algorithm utilizes the rest of the idle time slots in the leased resources to replicate workflow tasks. Therefore, the currently proposed algorithm applies deadline and reliability constraints. For the majority of studied scenarios, simulation experiments illustrate that the proposed algorithm increases the execution reliability of workflows while reducing the workflow execution cost.

With these challenges in mind, the present paper offers the following contributions:

- Replication-based fault-tolerance is considered a common approach for reliability enhancement. The current study proposes the Task Replication algorithm to increase workflow execution reliability at no extra cost. This algorithm utilizes the idle time slots of leased resources to replicate the proper tasks in those.
- Improving CbCP with Replication (ICR) is composed of three algorithms: Scheduling, Fix Up, and Task Replication. Scheduling is responsible for production of a preliminary schedule map of tasks of the workflow, Fix Up moves tasks of each time slot to an earlier position, if possible, to produce bigger idle time slots, and with larger ones, Task Replication receives a better chance to improve reliability.
- Experimental outcomes on real and randomly induced applications at various scales and heterogeneity degrees confirm that the proposed method can increase the workflow execution reliability while minimizing workflow execution costs.

The remaining structure of the current paper is as follows. Section 2 discusses related works. Section 3 describes the modeling of the system, problem definition, and evaluation metrics. Section 4 provides the details of the proposed algorithm and presents an example with its complexity and illustration. Section 5 presents the simulation results while Sect. 6 concludes the paper and suggests future work.

2 Related work

The authors of [20] divide workflow scheduling algorithms into two main areas: QoS-constrained and QoS optimization. The aim of QoS-constrained algorithms is to optimize some QoS parameters while meeting other user-defined QoS constraints. For instance, some studies have considered budget constraints while minimizing the makespan [21]. constructs a primitive scheduling that minimizes the makespan. If the cost falls within the budget, the schedule is finalized. Otherwise, tasks are remapped to less expensive resources that observe the cost limitations. The QoS optimization algorithm sets out to improve all QoS parameters. Towards this end, there have been investigations to determine a correspondence in QoS parameter relations, such as for time and cost in [22, 23]. The above-mentioned works assume that resources are always available. Nonetheless, in the actual world, the failure of resources and networks is unavoidable. There may be depletions in resources because of issues like link failure, power variation, and software/hardware failures [24]. Consequently, it is essential to consider reliability for efficient workflow scheduling and so reduce the failure of workflow execution.

Based on the common exponential distribution assumption in reliability research [25], for each processor, the occurrence of failures follows the Poisson distribution with failure rate (λ), which is a positive real number identical to the expected number of failures occurring in time unit t. Consequently, reliability throughout the interval of time t is $e^{-\lambda t}$. To achieve maximum system reliability while satisfying a given time constraint, [26] developed the Minimum Cost Match Schedule (MCMS) and Progressive Reliability Maximization Schedule (PRMS). Clearly, the discussed works attain a limited reliability and so special schemes are necessary, such as active replication. The primary and backup scheduling algorithm can tolerate one failure in the systems. The main representative methods include the efficient fault-tolerant reliability cost driven (eFRCD) [27], efficient fault-tolerant reliability driven (eFRD) [15], and minimum completion time with less replication cost (MCT-LRC) [18] algorithms. Regarding their limitations, these approaches first assume that no more than one failure happens at any one moment. In [14], Benoit et al. present the fault-tolerant scheduling algorithm (FTSA) which employs $\varepsilon + 1$ replicas for each task in order to guarantee system reliability. For a parallel application on heterogeneous systems based on the active replication scheme, Benoit et al. Go on to design a new scheduling algorithm in [21] that minimizes the schedule length under both throughput and reliability constraints. The main problem in [14, 28] is needing ε backups with high redundancy for each task so as to satisfy the application's reliability requirement. This, however, leads to large resource redundancy which adversely impacts execution by the system and incurs steep costs in resources. To satisfy an application's reliability requirement, recent studies have begun to explore active redundancy for each task approach [5, 29]. Active redundancy signifies that different tasks have different numbers of replicas. This approach has a lower resource cost than that of the fixed ε backups for each task based on active replication [29]. [5] and [30] propose fault-tolerant scheduling algorithms that feature reliability specifications to reduce resource redundancy (RR) and a quantitative fault-tolerant scheduling algorithm (QFEC+) with low execution charges, respectively. Both algorithms incorporate reliability analysis and active replication and employ a dynamic number of backups for different tasks by considering each task's sub-reliability requirement. Determining task sub-reliability demands is a RR and

QFEC+ limitation. In [5], the proposal of the DRR algorithm takes advantage of the deadline demand of a parallel application in RR.

The goal of the current paper is satisfying the demands of reliability and deadlines. However, some of the discussed algorithms do not consider a network's task communication and link failures.

The authors of [19] propose an algorithm for the cost-optimized, deadline, and reliability-constrained execution of workflows in Clouds. Dividing the workflow into a number of Clusters based on the Critical Parent (CbCP) concept. It is assumed that decreasing the workflow makespan is of no benefit for the user. Therefore, to minimize the execution cost, the CbCP algorithm utilizes almost all the time available before the deadline. Therefore, CbCP attempts to postpone the start time of tasks as much as possible, which consequently, may cause an increase in the workflow execution cost. Inspired by CbCP, the current paper's algorithm is saving workflow execution cost and time by inspecting the feasibility of beginning each task earlier in the leased resource, with no additional cost. Moreover, to increase workflow execution reliability, the proposed algorithm considers task replication as a way to increase workflow execution reliability. These replicas also increase resource utilization without incurring any extra costs.

3 System model and problem definition

This section discusses workflow and Cloud modeling and then outlines the problem.

3.1 Application and Cloud models

One of the best formats for the programming of scientific applications on distributed infrastructures is the workflow model, such as the Cloud and grid. The scheduling of scientific workflows in the Cloud is the focus of the current study's scheduling algorithm.

A scientific workflow utilizes the application model of a directed acyclic graph (DAG), because workflow tasks are optionally interconnected [31]. $G_\tau = (V, E, ET, CM)$ shall be the graph corresponding to a workflow in which $V = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is the set of tasks. The set of edges expressing the precedence relationships between tasks is E. Edge $(e_i, e_k) \in E$ signifies that τ_i is a parent of τ_k. As a single entry and a single exit task are required by the proposed algorithm, the start and the end of the workflow assume one dummy entry task and one dummy exit task, respectively. τ_{entry} and τ_{exit} represent these dummy tasks and feature zero processing time and zero communication. ET is an $n \times m$ matrix in which $exe_{i,j}$ signifies that the execution time of τ_i takes place on vm_j. With the cloud being naturally heterogeneous, the task computation time varies according to the resource. CM is an $n \times n$ matrix in which $cm_{i,k}$ represents the quantity of data moved from τ_i to τ_k. The amount of data communication between tasks is set and specified beforehand. A task can commence execution in a workflow upon the completion of its parents' executions and the transfer of its essential data.

The present work models the Cloud resource by $G_{rs} = (VM, \Lambda, PR)$ for scheduling algorithm, where VM indicates the finite set of m heterogeneous virtual machines: $VM = \{vm_1, vm_2, \ldots, vm_m\}$. $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ represents the failure rate of vm_j, which is a positive real number equal to the expected quantity of failures occurring in time unit t. $PR = \{pr_1, pr_2, \ldots, pr_m\}$ signifies the price of each resource per time unit.

Figure 1 depicts a sample workflow of ten tasks, τ_1 to τ_{10}. While a single entry and single exit task are required by the proposed algorithm, dummy tasks τ_{entry} and τ_{exit} are determined. Figure 1 shows each weighted edge, which represents both the estimated data transfer time and the precedence constraint among the corresponding tasks. Assumed for each task τ_i are three different possible resources, i.e., vm_1, vm_2, and vm_3, which use a different QoS to execute the task. Table 1 presents the task execution times on various resources (ET), each resource per time unit (PR) cost, and each resource's failure rate (Λ), respectively. For the commencement of τ_5's processing, all the data required by τ_2 and τ_3 must be sent. This example shows that the workflow makespan is the end task's finish time, τ_{10}.

3.2 Problem definition

According to Sect. 1, it is impossible for the reliability of a workflow execution to be 100%. However, a workflow execution may be considered reliable if the system can meet the workflow's reliability demand. The present study's problem may be described by the following. The scheduling system's input contains the given workflow G_τ, deadline D, and reliability R. The aim is to locate a map of tasks used by resources to produce a concrete schedule so that the workflow execution $Cost$ is minimized whilst the makespan meets the deadline and the workflow's acquired reliability, $Rel_{schedul}$, satisfies the requirements. Eq. (1) provides the program's formal descriptions and Table 2 lists the present study's important notations and their definitions.

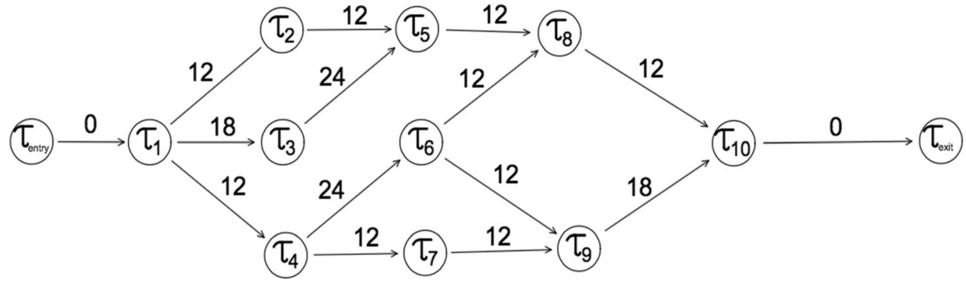


Fig. 1 A sample workflow [19]

Table 1 Available resources for the workflow of Fig. 1

VM	ET										PR	Λ
	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}		
vm_1	12	30	18	24	18	24	30	18	30	12	3$	0.00010
vm_2	30	72	30	36	48	48	48	36	48	30	2$	0.00015
vm_3	48	96	54	60	66	66	66	48	84	48	1$	0.00018

$$\begin{aligned} MinCost &= \left[\sum_{x=1}^{y} \frac{FT_{cl_x,j} - ST_{cl_x,j}}{lenghtofinterval}\right] \times pr_j \\ M &= FT_{\tau_{exit}} - ST_{\tau_{entry}} \le D \\ Rel_{schedul} &\ge R \end{aligned} \tag{1}$$

Cost is what the user pays for the execution of the workflow by the resource and for network consumption. As discussed, a clustering algorithm is the basis of the current research. That is to say, in the Critical Parent view, the workflow is divided into clusters in which y is the quantity of generated clusters; $FT_{cl_x,j} - ST_{cl_x,j}$ is the total execution time of cluster (cl_x) on resource vm_j; $Rel_{schedul}$ is the workflow execution reliability or the workflow execution's probability of being successfully completed; and M is the workflow makespan which signifies the time required to finish all the tasks' executions. Finally, during scheduling, the start time and finish time of each task are also specified.

A pricing model popular among most commercial infrastructure as a service (IaaS) Cloud service providers models the resource usage cost, which is founded on a pay-as-you-go plan. This model calculates charges by the number of time intervals the resource is used, even if the last one is not completely consumed.

To model reliability, there are two major kinds of failures: transient failure (also called random hardware failure) and permanent failure. Once a permanent failure happens, the resource cannot be restored unless by replacement. On the other hand, in the case of the most probable consequence of transient failure, the failure lasts a short time and finishes without damaging the resources [25]. Consequently, the present study mostly considers transient

Table 2 Important notations in this study

Notation	Definition
$exe_{i,j}$	Execution time of task τ_i. on vm_j
$cm_{i,k}$	Communication time between the tasks τ_i. and τ_k
λ_j	Constant failure rate of vm_j. per time unit
pr_j	Price of each resource per time unit
$TET_{cl_x,j}$	Total Execution Time of cluster x (cl_x) on vm_j
$CMR_{k,i}$	Communication Reliability between parent and its child tasks
$CR_{i,j}$	Computation Reliability of task τ_i on the vm_j
$CR_{cl_x,j}$	Computation Reliability of cluster x (cl_x) on the vm_j
CP_i	Critical Parent of task τ_i
$Rel_{schedul}$	liability of workflow execution
$Rel_{min}^{cl_x}$	Minimum sub-reliability requirement for the cluster x (cl_x)
$Rel_{Actual}^{cl_{x,j}}$	Actual reliability achieved by allocations of cluster x (cl_x) on vm_j
Rel_j^i	Reliability attained by scheduling a task τ_i on vm_j
RF_i	Failure probability for task τ_i on vm_j
RF_{slot_w}	Probability of failure of an idle time slot on vm_j when task τ_i is assigned

failures. Generally, the occurrence of a transient failure for a task in a DAG-based application follows the Poisson distribution [5, 16, 32]. Therefore, the reliability of an event in time unit t is indicated by Eq. (2):

$$R(t) = e^{-\lambda t} \tag{2}$$

The present paper also addresses heterogeneous systems made up of various hardware and software with different configurations or capacities. As a result, each resource's Mean Time Between Failures (MTBF) also differs from each other. Like [32], the current research presumes that failures do not happen during processor times.

4 The scheduling algorithm

The aim of the present paper is the QoS-constrained static scheduling of scientific workflows on the IaaS Cloud platform. Because of its data communication and precedence limitations, there is a challenge in optimally scheduling a workflow. As reliability is related to Communication Reliability (CMR) and Computation Reliability (CR), the scheduling issue becomes more difficult to solve.

$$Rel_{schedul} = \prod_{i=1}^{n} CR_{i,j} \times CMR_{k,i}, \tau_i \in V, (e_k, e_i) \in E, vm_j \in VM \tag{3}$$

CMR is the probability of the successful transfer of a parent task generated message, for example τ_k, to processors located with the child tasks, for instance τ_i. CR is the probability of τ_i being successfully executed on vm_j. CbCP holds that the workflow structure increases reliability. In other words, the obtained reliability of the workflow is enhanced by lowering the quantity of messages sent from parent tasks to child tasks. Furthermore, replication is the most widely used mechanism for enhancing the reliability of services. Replication can be achieved by either redundancy in time (task resubmission) or redundancy in space (task duplication). The rationale behind task replication with ε number of replicas is that $(\varepsilon - 1)$ failures can be tolerated without affecting the workflow makespan. The downside of task replication is the consumption of extra resources. In order to mitigate such effects and save on costs, the current research proposes a scheduling algorithm which employs the idle time slots of leased resources to replicate tasks. Therefore, the objective of the proposed ICR (Improving CbCP with Replication) algorithm is raising the execution reliability of workflows within a user-specified deadline and reliability in a public Cloud environment.

This section first discusses the main concepts and then provides some basic definitions. The ICR scheduling algorithm is then elaborated on and its time complexity computed. Finally, the algorithm's operation is demonstrated through an illustrative example.

4.1 Main ideas

The proposed algorithm completes three different steps at a high level:

Step 1 *Allocation of Cloud resources and task scheduling* This step involves obtaining the suboptimal number and the type of resources able to fulfill the workflow within its deadline and reliability constraints.

Step 2 *Fix Up* This second step attempts to explore possibility of starting each task earlier in the leased resource, without any extra cost, by shifting each task toward the beginning of the time interval.

Step 3 *Task replication* The third step determines the placement of idle time slots and the replication of a proper tasks in those for enhancing workflow execution reliability without incurring any extra cost. The next sub-sections discuss each step-in detail.

4.2 Basic definitions

In its ICR scheduling algorithm, the present paper intends to find, for all tasks, the priority and Critical Parent. Consequently, calculating some workflow task parameters is necessary before scheduling the workflow proposed by [19].

A task's priority is the length of the longest path from the task to τ_{exit}. When calculating the length of the path, the communication times are excluded and solely the computation times are considered.

$$\begin{aligned} &Rank_i = \overline{exe_{i,j}}, && if\ succ_i = \varphi,\quad \tau_i \in V, vm_j\ VM \\ &Rank_i = \max_{\tau_k \in\ succ_i} (Rank_k + \overline{exe_{i,j}}), && Otherwise \end{aligned} \tag{4}$$

Calculation of the rank is repeatedly performed by passing over the task upwardly, starting from τ_{exit}. In Eq. (4), the average computation time of task τ_i is $\overline{exe}_{i,j}$ and the set of immediate successor tasks of τ_i is $succ_i$.

CP_i is the Critical Parent of τ_i, whose sum of finish time and data transfer time is greater than that of all other parents.

$$CP_i = k \in pred_i \; s.t \; EFT_k + cm_{k,i} \; > \; EFT_l + cm_{l,i} where \; l \in pred_i \; and \; k \neq l. \tag{5}$$

For each unscheduled task τ_i, its Earliest Start Time (EST_i) is the earliest time at which τ_i can start its computation. Obviously, it is impossible to compute the precise EST_i, since a Cloud is a heterogeneous environment and the computation time of tasks varies from resource to resource. Thus, the current work must approximate the execution and data transmission time for each unscheduled task. Between the available approximation alternatives (e.g., the average, the median, or the minimum), the minimum execution and data transmission time are preferred. Therefore, the Earliest Start Time (EST_i) is defined as follows:

$$EST_i = 0, \textit{for the entry node}$$
$$EST_i = max_{k \in pred_i}(EFT_{k\, where\, \tau_i \,\&\, \tau_k \in identical\, vm} \; \| \; EFT_{k\, where\, \tau_i \& \tau_k \notin identical\, vm} + cm_{k,i}) \tag{6}$$

In addition, defined for each unscheduled task τ_i is its Earliest Finish Time (EFT_i) or the earliest time at which τ_i can finish its computation. Once again, it is impossible to compute EFT_i exactly and it must be computed according to the approximate execution and data transmission time as follows:

$$EFT_i = EST_i + exe_{i,j} \tag{7}$$

Similarly, the Latest Finish Time (LFT_i) or the latest time in which τ_i can complete its computation, can be defined as:

$$LFT_i = Deadline, \textit{For the exit node}$$
$$LFT_i = min_{k \in succ_i}(LST_{k\, where\, \tau_i \,\&\, \tau_k \in identical\, vm} \; \| \; LST_{k\, where\, \tau_i \,\&\, \tau_k \notin identical\, vm} - cm_{i,k}) \tag{8}$$

Finally, (9) shows the Latest Start Time (LST_i) which is the latest time at which τ_i can start its computation:

$$LST_i = LFT_i - exe_{i,j} \tag{9}$$

4.3 Allocating and scheduling

Among the existing approaches for scheduling workflow applications in public Cloud environments, the CbCP (Clustering based on Critical Parent) algorithm s works with assumptions closest to those of the system and application models discussed in Sect. 3. CbCP is a cost minimizer with a deadline- and reliability-constrained algorithm that operates by assigning all the tasks of a workflow cluster to the same resource.

The first step of the ICR algorithm includes the determination of ordering and placing each task of clusters according to CbCP. The resource assignment algorithm then determines the quantity and kind of VMs to be utilized based on the clustering algorithm for workflow execution. At the beginning, the task priority should be computed before clustering. Similar to the state-of- the-art study by [33], the present research ranks tasks according to their priority which are based on upward ranks. The length of the longest path from τ_i to τ_{exit} is task τ_i's priority. In figuring a path's priority or path length, communication times are neglected and solely computation times are counted (Eq. 4). The priority of τ_{entry} is the sum of computation costs along the longest path. This schedule length can never be lower than the priority of DAG's τ_{entry}.

Algorithm 1. The ICR Scheduling Algorithm

procedure ScheduleWorkflow ($\boldsymbol{G_\tau(V, E, ET, CM)}$, $\boldsymbol{D}$, $\boldsymbol{R}$)
determine available virtual machines
add $\boldsymbol{\tau_{entry}}$, $\boldsymbol{\tau_{exit}}$ and their corresponding edges to G
Compute $\boldsymbol{Rank_i}$ for each task according to Eq. 4, starting from the $\boldsymbol{\tau_{exit}}$
compute $\boldsymbol{EST_i}$ for each task according to Eq. 6
compute $\boldsymbol{EFT_i}$ for each task according to Eq. 7
compute $\boldsymbol{LFT_i}$ for each task according to Eq. 8
compute $\boldsymbol{LST_i}$ for each task according to Eq. 9
compute $\boldsymbol{CP_i}$ for each task in G according to Eq. 5
$EST(\boldsymbol{\tau_{entry}}) \leftarrow 0$, $LFT(\boldsymbol{\tau_{exit}}) \leftarrow$ D
call clustering ($\boldsymbol{G_\tau}$)
call Resource Assignment ($\boldsymbol{G_\tau}$)
call Fix Up ($\boldsymbol{G_\tau}$)
call Task Replication ($\boldsymbol{G_\tau}$)
end procedure

The clustering algorithm is based on CbCP. In other words, for each τ_i, a Critical Parent (CP_i) is computed, which signifies the assignment of both the task and its Critical Parent to the same vm. Task graph nodes are sorted by the lowest rank first order on the list. Cluster generation begins from the first task on the list not yet assigned to a cluster. The search traces the selected task any successor task's Critical Parent. Then, the current task is added to the appropriate cluster. If not, a new cluster is generated for this task. The generation of a cluster is accomplished whenever there are no more unassigned tasks on the list.

For assigning clusters to VM, the present work adopts the method from the CbCP algorithm which examines available VM. In order to find a vm_j able to execute each cluster task before its LFT and by satisfying its sub-reliability, available VM are examined, starting from the cheapest to the more expensive. The first vm_j able to meet this requirement at the cheapest price is selected. Let R be the reliability demand and y be the quantity of generated clusters. Therefore, the minimum sub-reliability demand for the cluster with the highest priority is computed as:

$$Rel_{min}^{cl_x} = \frac{R}{1 \times \prod_{i=x+1}^{y} Rel_{max}^{cl_{i,j}}}, x = 1$$
$$Rel_{min}^{cl_x} = \frac{R}{\prod_{i=1}^{x-1} Rel_{Actual}^{cl_{i,j}} \times \prod_{i=x+1}^{y} Rel_{max}^{cl_{i,j}}}, 2 \le x \le y, vm_j \in VM \tag{10}$$

Let D be the overall deadline requirement of an application. While a workflow consists of several tasks, each task is associated with a sub-deadline related to the overall deadline. Thus, the sub-deadlines of exit tasks are equal to D and the sub-deadlines of the rest of the tasks are calculated on the authority of a traversal of the workflow graph in reverse topological order. This policy attempts to find the cheapest schedule able to execute each workflow task before its Latest Finish Time (LFT).

4.4 Fix up

As previously mentioned, CbCP attempts to postpone the start time of tasks as much as possible. According to Eqs. (11, 12, 13), this algorithm computes the sub-deadline of each task based on the Latest Finish Time (LFT) in order to satisfy the user-defined deadline. Suppose the Actual Start Time (AST) of task τ_i is AST_i.

$$AST_i = LFT_i - exe_{i,j} \tag{11}$$

$$LFT_i = Deadline, \quad For\ the\ exit\ node \tag{12}$$

$$LFT_k = \min\left(AST_{i \in succ_k where \tau_i \& \tau_k \in identicalvm} \;\|\; AST_{i \in succ_k where \tau_i \& \tau_k \notin identicalvm} - cm_{k,i}\right), unassignedpred_i \tag{13}$$

Therefore, the Fix Up algorithm attempts to inspect feasibility of starting each task earlier in the leased resource, without any extra cost, by shifting each task toward the beginning of the time intervals in an iterative process. In order to shift task τ_i, the present paper proposes new factors for tasks: Definite Start Time (DST) and Definite Finish Time (DFT). Algorithm 2 describes the heuristic Fix Up.

Algorithm 2 sorts the nodes of the task graph by the highest rank first order on the list (Line 2). The determination of a DST is originated from the first task on the list. As stated in Eq. (14), the DST_{entry} is zero. For each task τ_i, the current work computes its Definite Start Time, DST_i, as the exact time at which τ_i start its computation (Line 5). Moreover, for each task τ_i, the exact time at which τ_i fulfill its computation is described as its Definite Finish Time DFT_i (line 8). Therefore,DST_i and DFT_i are computed as follows:

Algorithm 2. Fix Up Algorithm

1. ***procedure*** *Fix Up* (G_τ)
2. *sort the tasks in a list by descending order of rank value*
3. ***while*** *(there exists an unchecked task)* ***do***
4. *Select the first task* τ_i *from the list*
5. *Compute the* DST_i *of the task,* according to Eq. 14
6. ***if*** $DST_i < AST_i$ ***then***
7. *shift task* τ_i *to* DST_i
8. *Compute the* DFT_i *of the task,* according to Eq. 15
9. *remove* τ_i *from the list*
10. ***else***
11. $DST_i = AST_i$
12. $DFT_i = AFT_i$
13. *remove* τ_i *from the list*
14. ***end if***
15. ***end while***
16. ***end procedure***

$$DST_i = 0, \quad Fortheentrynode \tag{14}$$

$$DST_i = max\left(DFT_{k \in pred_i\, where\, \tau_i \,\&\, \tau_k \in identical\, vm} \,\|\, DFT_{k \in pred_i\, where\, \tau_i \,\&\, \tau_k \notin identical\, vm} + cm_{k,i}\right)$$
$$DFT_i = DST_i + exe_{i,j} \tag{15}$$

The algorithm compares a task's AST_i and DST_i. If DST_i is smaller than AST_i, then task τ_i is shifted to DST_i and DFT_i is updated. Otherwise, just DST_i and DFT_i are updated (Lines 6–13). The Fix Up algorithm is accomplished whenever there are no unchecked tasks on the list.

4.5 Task replication

The ICR algorithm attempts to utilize accessible free time gaps in the schedule map for task replication in the iterative replication process. In order to replicate task τ_i to a corresponding *vm*, the present paper proposes new factors for tasks and idle time slots called risk factors. Algorithm 3 describes the heuristic algorithm of Task Replication.

Let RF_i indicate the failure probability for task τ_i on vm_j. Since a workflow consists of a number of tasks, each task is associated with RF_i according to the reliability of its *vm* and the duration of its execution time. Therefore, the RF_i of each task is computed according to Eq. (16). Suppose that the reliability attained by scheduling task τ_i on vm_j is Rel^i_j.

$$RF_i = \left(1 - Rel^i_j\right) \times exe_{i,j} \tag{16}$$

$$Rel^i_j = e^{-\lambda_j \times \left(\left(\sum_{\tau_k \in vm_j} exe_{k,j}\right) + exe_{i,j}\right)} \tag{17}$$

The Task Replication algorithm prioritizes tasks into a decreasing order of risk factor.

Let RF_{slot_w} be the probability of failure of a time slot on vm_j when task τ_i is assigned. RF_{slot_w} depends on three items: the type of *vm*, the task assigned to that *vm*, and the communication reliability. As a Cloud is a heterogeneous environment, the probability of failure for each *vm*, computation time, and reliability of tasks vary from one *vm* to another. In addition, based on Eq. (3), CMR, which represents the communication reliability link between two resources, affects the amount of reliability achievement. Therefore, the location of both the parent tasks and child tasks is critical.

Algorithm 3. Task Replication Algorithm

1. ***procedure*** *replication(G_τ)*
2. *sort the idle time slots into ascending order of size*
3. *sort the tasks by decreasing order of Risk Factor* according to Eq. 16
4. ***while*** *(there exists an unchecked task)* ***do***
5. *Select the first task τ_i from the list for replication*
6. *Compute the RF_{slot_w} of the task, on all available idle time slots,* according to Eq. 18
7. ***if*** *τ_i fits any processor* ***then***
8. *select processor rs_j with least Risk factor and create replica τ_i*
9. *Remove τ_i from the list*
10. *Update idle time slots information*
11. *Update workflow execution reliability according to Eq. 20*
12. ***else***
13. *remove τ_i from the list*
14. ***end if***
15. ***end while***
16. ***end procedure***

In Line 14, ICR calls upon the Task Replication algorithm to obtain a list of all possible idle time slots useful for task replication purposes. Idle time slots exist in scheduling of workflows because of two reasons. The first is dependencies among tasks which may lead to periods in which the next task scheduled to a *vm* must wait for data that are being generated during the execution of another task in another *vm*. The second reason for idle time slots in scheduling of workflow is that some paid periods are only partially used in some situations. These idle time slots are sorted into ascending order of size.

Lines 4–14 define an order for the tentative replication of tasks in available idle time slots. Tasks are sorted according to their risk factor criteria, as follows:

Computation reliability (CR) and communication reliability (CMR) are calculated from tables of reliability provided by the Cloud provider.

$$RF_{slot_w} = \left(1 - CR_{i,j}\right) \times exe_{i,j} \times \left(1 - CMR_{k,i}\right) \tag{18}$$

For task replication, the current study employs the following steps to select the task and the corresponding processor.

Compute the RF_{slot_w} of the first task on the list of all available idle time slots. In other words, if task τ_i has been assigned to a vm, then this vm is either unavailable or available for τ_i. ICR then replicates task τ_i with a maximum risk factor on corresponding vm_j with a minimum risk factor that generates the maximum workflow execution reliability value. A task is considered as assigned to a slot if its execution does not violate the end of the time interval, is completed before its sub-deadline, and does not violate task dependencies; that is, in this slot, the task does not run before a predecessor or after a successor. As the objective of this replication is to increase reliability rather than fault tolerance, it should be noted that space replication is the target of ICR.

Because of the reliability enhancement of τ_i, the workflow execution reliability is then improved and is changed to:

$$Rel^i_{new} = 1 - \left(\left(1 - Rel^i_j\right) \times \left(1 - Rel^i_z\right)\right) vm_j, vm_z \in VM \tag{19}$$

$$Rel_{schedul} = Rel^i_{new} \times \frac{\prod_{x=1}^{y} Rel^{cl_{x,j}}_{Actual}}{Rel^i_j} \tag{20}$$

As previously mentioned, $Rel^{cl_{x,j}}_{Actual}$ is the probability that cl_x is successfully executed on vm_j.

4.6 Time complexity

To determine the time complexity of the proposed algorithm, suppose that the schedule workflow receives workflow $G_\tau = (V, E)$ as an input with n tasks and e edges. Additionally, assume that the maximum quantity of resource kinds for each task is m while $|S|$ is the number of idle time slots. As G_τ is a directed acyclic graph, the maximum number of edges can be assumed as $O(V)^2$. Therefore, the present work first computes the time complexity of the algorithm's principal sections as follows:

Firstly, the algorithm traverses each task of the task graph and calculates the start times and completion times. At each node, the incoming and outgoing edges are considered and, in the worst case, all of the DAG edges need to be checked. Therefore, the worst-case complexity of these stages is $O(|E|)$, while $|E|$ is the quantity of edges.

Because the array queue must be generated, this can be performed at time $O(|V|log|V|)$, which is the time for sorting DAG nodes in an ascending order of importance [34].

For the resource assignment algorithm, each cluster's resources must be evaluated so as to detect the cheapest one which also respects both the sub-reliability and sub-deadline constraints. Each resource evaluation should compute the actual start time of the task on that resource, which requires considering all parent tasks and their edges. In the worst case, a node has $n-1$ unassigned predecessors, so the time complexity of updating the LFT for all nodes is $O(|V|)$.

The Fix Up algorithm traverses each task of the task graph and computes the Definite Start Times and Definite Finish Times. The worst-case complexity of these steps is $O(|E|)$, where $|E|$ is the number of edges.

The Task Replication algorithm traverses each task of the task graph and computes risk factor RF_i. The algorithm then tests this on all available idle time slots for each task in order to find the least RF_{slot_w} which respects both the task dependencies and sub-deadline constraints. In the worst case, the time complexity of replication for all nodes is $O(|V| \times |S|)$ while the maximum number of idle time slots can be assumed as $O(|V|)$.

Accordingly, the overall time complexity of the ICR algorithm is O(|V| +|E| +|V| log |V| +|E| +|V||V|). For a dense graph, the quantity of edges is proportional to O(|V|^2). Therefore, the worst-case complexity of the ICR algorithm is O(|V|^2).

4.7 An illustrative example

In order to show how the algorithm works, the current paper traces the algorithm's operation on the sample graph shown in Fig. 1 [19]. The graph consists of ten tasks, from τ_1 to τ_{10}, and two dummy tasks, τ_{entry} and τ_{exit}. There are three different types of resources for each task τ_i, i.e., vm_1, vm_2, and vm_3, which can execute the task with different QoS. Table 1 provides the execution times of tasks on different resources (ET), the price of each resource per time unit (PR), and the failure rate of each resource (Λ), respectively. In Fig. 1, the number above each edge shows the estimated data transfer time among the corresponding tasks. Finally, the overall deadline and reliability of the workflow are 300 and 0.94, respectively. As mentioned before, ICR involves three algorithms: Scheduling, Fix Up, and Task Replication. The first step of the ICR algorithm includes the determination of ordering and placing each task of clusters according to CbCP. In other words, with these settings, the CbCP algorithm, which is the state-of-the-art algorithm for scheduling workflows on Clouds, generates the schedule map illustrated in Fig. 2 [19]. The CbCP consists of four main algorithms: clustering, reliability distribution, deadline distribution, and resource assignment. The clustering algorithm is based on a critical parent. In the reliability distribution algorithm, the sub-reliability requirement for the clusters is continuously calculated based upon the actual reliability obtained by previous assignments, as well as presuppose reliability achieved by assigning unallocated clusters to the processor

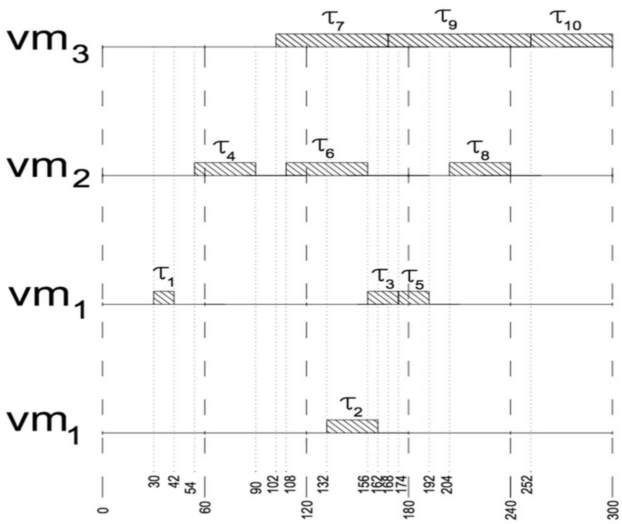


Fig. 2 Sample workflow schedule map

with maximum reliability to ensure that the overall reliability requirement is met. In the deadline distribution algorithm, the distribution of the overall workflow deadline over individual tasks is based on the Latest Finish Time. That is, if each task finishes before its sub-deadline, the whole workflow is completed before the user-defined deadline. Finally, the resource assignment chooses the cheapest service for each cluster while satisfying its sub-reliability and sub-deadline.

ICR then calls the Fix Up algorithm to compute the DST_i and DFT_i for each task τ_i. Based on Algorithm 2, the nodes of the task graph are sorted by the highest rank first order on the list. The array list for this DAG is: $RankList = \{\tau_1, \tau_2, \tau_4, \tau_3, \tau_7, \tau_6, \tau_5, \tau_9, \tau_8, \tau_{10}\}$. The computation of a DST is started from the first task on the list. For example, assume $DST_1 = 0$ and $DFT_1 = 12$. As the value of DST_1 is smaller than that of AST_1, the algorithm performs a shift. The Fix Up algorithm is completed when there are no more unchecked tasks on the list. After the possible shifting by the Fix Up algorithm in Fig. 2, the result is the schedule map of the workflow as seen in Fig. 3.

Finally, ICR calls upon the Task Replication algorithm, i.e., Algorithm 3, for the sample workflow in Fig. 1. Based on Algorithm 3, the risk factor is computed for each task of the workflow according to Eq. (16). The generation of a replication is started from the first task on the Risk Factor List, which has not yet been checked by any resource. The tasks on the list are sorted by descending order of risk factors. Thus, the array list for this DAG is: $Risk\ Factor\ List = \{\tau_{10}, \tau_7, \tau_8, \tau_6, \tau_9, \tau_4, \tau_2, \tau_3, \tau_1, \tau_5\}$. τ_{10} appears as the first task on the list. When the suitability of any idle time slots is evaluated for τ_{10}, all are considered invalid. This is due to, the DFT of τ_8 and τ_9, the predecessors of τ_{10}, are 144 and 222, respectively. Therefore, if τ_{10} is applied in vm_2, it must be executed for an extra time interval. The condition of the two next tasks, τ_7 and τ_8, is the same as that of τ_{10}. However, τ_6 does not violate tasks dependencies and is completed before the end of the time interval. Thus, τ_6 is chosen to be replicated to vm_1. The rest of the replications are generated by following this process.

Figure 4 depicts the scheduling of the workflow in Fig. 1 with task replication, which makes it possible to meet the application deadline and reliability. The hatched boxes indicate the tasks, the crossed boxes signify the replica of a task, and the numbers outside the boxes signify the task numbers. The execution start time and finish time of each task can be determined according to the time bar. τ_{entry} and τ_{exit} with zero computation and zero communication time have no effect on the scheduling method and are not appeared in the schedule map.

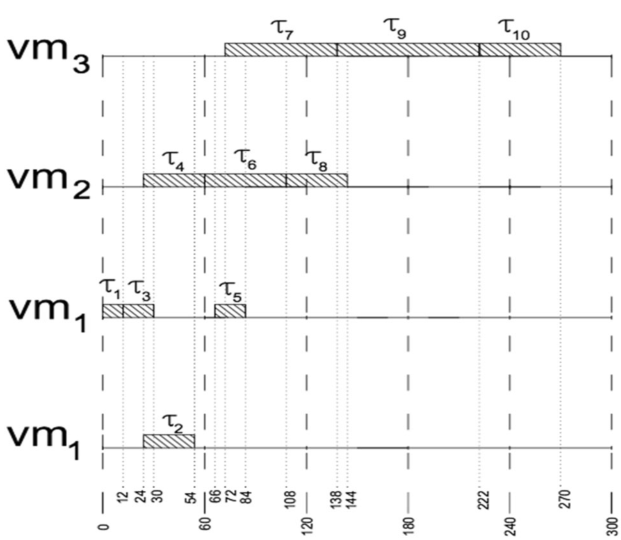


Fig. 3 Sample workflow schedule map after the Fix UP algorithm

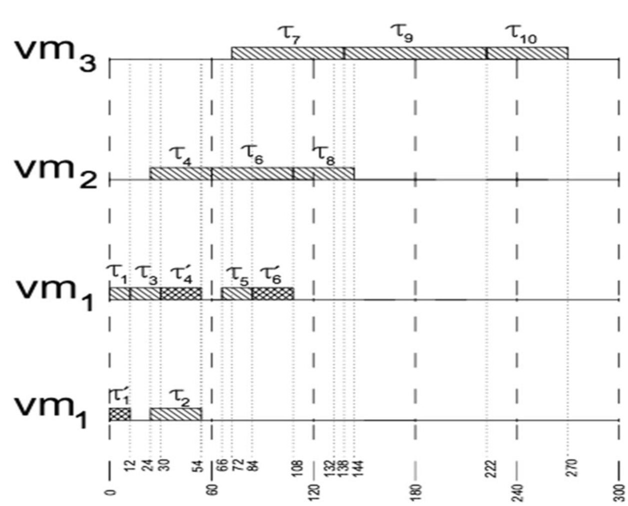


Fig. 4 Sample workflow schedule map after the Replication algorithm

As discussed above, the Fix Up algorithm attempts to explore possibility of starting each task earlier in the leased resource, without imposing any extra cost, by shifting each task toward the beginning of the time interval. Moreover, in the application of the Task Replication algorithm to idle time slots, some tasks are replicated to increase the reliability of the workflow. According to the results, the total time, reliability, and cost are 270, 0.9581, and $19, respectively.

5 Performance evaluation

This section presents the results of the current work's simulations that utilize the improved CbCP with replication (ICR) algorithm.

5.1 Experimental workflows

To evaluate a workflow scheduling algorithm, its performance should be measured on some sample workflows. For instance, such an evaluation can employ a random DAG generator to construct different workflows with various properties or utilize a library of realistic workflows designed for the scientific or business community.

One of the introductory works in this area is by Bharathi et al. [35], who studied the framework of five realistic workflows from different scientific applications, namely CyberShake, Epigenomics, LIGO, and SIPHT. These graphs are based on real scientific workflows in different fields of science, such as astronomy, earthquake science, biology, and gravitational physics. Figure 5 provides the approximate structure of these workflows which have a few number of nodes. It should be mentioned that these workflows have different structural characteristics in relation to their composition and key components, such as pipeline, data aggregation, data distribution, and data redistribution. For each workflow, tasks with similar color are in the identical class and can be processed with a common service.

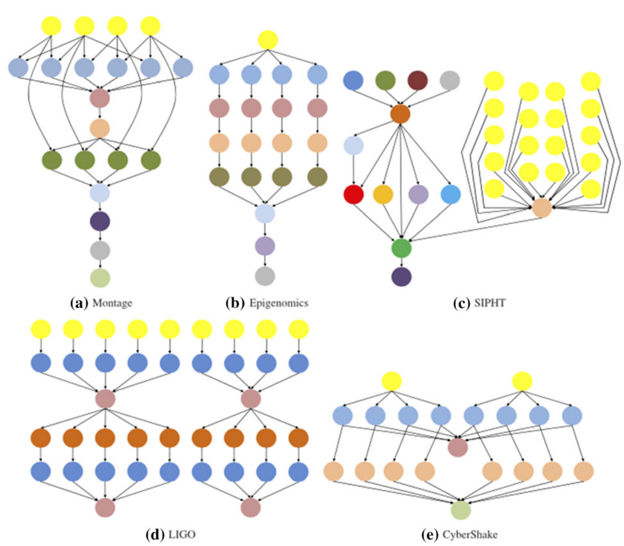


Fig. 5 The structure of five realistic scientific workflows [35]

Bharathi et al. provide a detailed specification for each workflow which explains their structures, data, and computational demands. The DAX (directed acyclic graph in XML) format of these workflows is accessible on their website. For its experiments, the present research chooses three sizes of workflows: small (about 25 tasks), medium (about 50 tasks), and large (about 100 tasks).

5.2 Experimental setup

The resource class determined in the current study is based upon Amazon AWS EC2. Table 3 presents the instances and their related leasing prices for a period of 60 min. The Pegasus Workflow Generator [35] generates the mentioned workflows.

The approaches of [19] and [30] are similar to the current work. The authors of [19] introduce the CbCP algorithm to minimize the cost of an application and to satisfy its deadline and reliability requirement on heterogeneous distributed systems. The limitation of CbCP is that, the algorithm does not consider the idle time slots in leased resources. In [30], the authors present the QFEC+ algorithm to minimize the execution cost to meet an application's reliability requirements. The two necessary limitations of the QFEC+ procedure are its ignoring of the workflow structure and the high sub-reliability requirements of the tasks.

To compare the current study's simulation results with those from the CbCP and QFEC+ algorithms, five different deadline intervals and reliability values are specified, from tight to relaxed. At first, for each workflow, the HEFT strategy calculates the deadline, since it must be greater than or equal to the makespan of scheduling the same workflow with the HEFT strategy. Then, various deadline

Table 3 VM types used in the experiments

Type	Memory (GB)	Core speed (ECU)	Cores	Cost ($)
m1.small	1.7	1	1	0.06
mi.medium	3.75	2	2	0.12
m1.large	7.5	2	2	0.24
m1.xlarge	15	2	2	0.48
m3.xlarge	15	3.25	3.25	0.50
m3.xxlarge	30	3.25	3.25	1.00

thresholds are computed by multiplying the deadline by the constant c, where c ranges from 1 to 5. When $c = 1$, the deadline is very tight, while higher values of c represent more relaxed deadlines. Moreover, different reliability thresholds are defined from 0.95 to 0.9 with 0.01 decrements. It is assumed that the reliability of the processors and the communication links utilized in this stage are predefined. That is to say, the Cloud provider has obtained these values.

5.3 Experimental results

To evaluate its ICR scheduling algorithm, the present paper must assign a deadline and reliability to each workflow. In order to calculate each workflow application's deadline and reliability, their execution is simulated with HEFT [33]. Moreover, to solve the difference in the attribution of workflows, the total cost is normalized to make the comparison easier. As a result, the normalized cost (NC) of a workflow is computed by dividing the current execution cost of the workflow by the execution cost of the cheapest possible schedule.

To assess the ICR, CbCP, and QFEC+ algorithms by testing, cost and system reliability are selected for the evaluation criteria. The schedule cost compares the monetary cost among the algorithms, while system reliability measures the performance of these algorithms.

In the current study's simulation experiments, all three algorithms successfully scheduled all workflows before their deadlines according to the reliability requirements for tight deadlines (short deadline factor) and the workflow reliability threshold R which is equal to 0.95. The system reliability of the three algorithms is compared with respect to graph specifications. It should be noted that CyberShake, Epigenomics, LIGO, Montage and SIPHT workflows have different structural characteristics in relation to their composition and key components, such as pipeline, data aggregation, data distribution, and data redistribution. Figure 6 presents the overall experimental results. As mentioned, the limitation of CbCP is not considering idle time slots to increase the overall reliability, while the QFEC+ algorithm's key limitations are ignoring the workflow structure, as well as the high sub-reliability requirements of all tasks. Compared to CbCP and QFEC+ , ICR utilizes the idle time slots of allocated resources to replicate proper tasks in, which is thought to enhance the workflow execution reliability. It should be mentioned that the degree of enhancing reliability varies among different workflows as this depends on the workflow structure and size. For instance, in the cases of Epigenomics and LIGO for small and medium workflow reliability, improvement is impossible because the number of idle time slots is very low in these cases. In contrast, the ICR algorithm achieves noticeable improvement in system reliability with Montage, CyberShake, and SIPHT for small and medium workflows, in which the amount of replication is more than in other types of workflows. In other words, ICR algorithm achieves noticeable improvement in system reliability in the workflows with high interdependencies among tasks, i.e., CyberShake and Montage workflows. In these types of workflows, the quantity of idle time slots exist in scheduling is more. The reason is that the dependencies among tasks located in different clusters are high, that may lead to periods, in which, the next task scheduled to a *vm* must wait for data that are being generated during the execution of another task in another *vm*. As a result, by increasing the number of idle time slots, the rate of replication is increased and influences system reliability. On the contrary, according to the structure of high-parallelism workflows such as Epigenomics, the number of idle time slots are low and the system reliability is just satisfied.

Figure 6 demonstrates that system reliability lowers with an increasing number of tasks, the reason for which is explained as follows. When the task number increases, the total execution time of each cluster correspondingly lengthens. Consequently, according to Eq. (2), system reliability falls. Moreover, improvement of the workflow's execution reliability declines by relaxing the deadline, since the possibility of selecting a slower resource via the resource assignment algorithm increases. Therefore, in this type of resource, the execution time of each task is longer than that of fast resources. As a result, the chance of tasks finishing execution in the idle time slots of leased resources decreases.

Figure 7 provides the cost of scheduling all workflows with the ICR, CbCP, and QFEC+ algorithms. The QFEC+ method iteratively selects available replicas and processors with minimum execution times for each task until its sub-reliability demand is satisfied. On the contrary, the CbCP algorithm uses almost all available deadlines to meet the deadline. In other words, CbCP attempts to postpone the start time of tasks as much as possible even if it may cause to increase the workflow execution cost. Thus, ICR employs the Fix Up algorithm to explore the possibility of starting each task earlier in the leased resource, without imposing any extra cost, by shifting each task toward the beginning of the time interval. In some cases, this process frees up a number of time intervals, for example, in the cases of Montage and CyberShake for small and medium workflow executions, in which the cost improvement is noticeable because of the structural properties of the workflows. In these types of workflows, the dependencies between tasks located in different clusters are high. As a result, this can generate idle time slots in leased resources which may affect the workflow execution cost. In contrast, in the cases of Epigenomics, SIPHT, and LIGO,

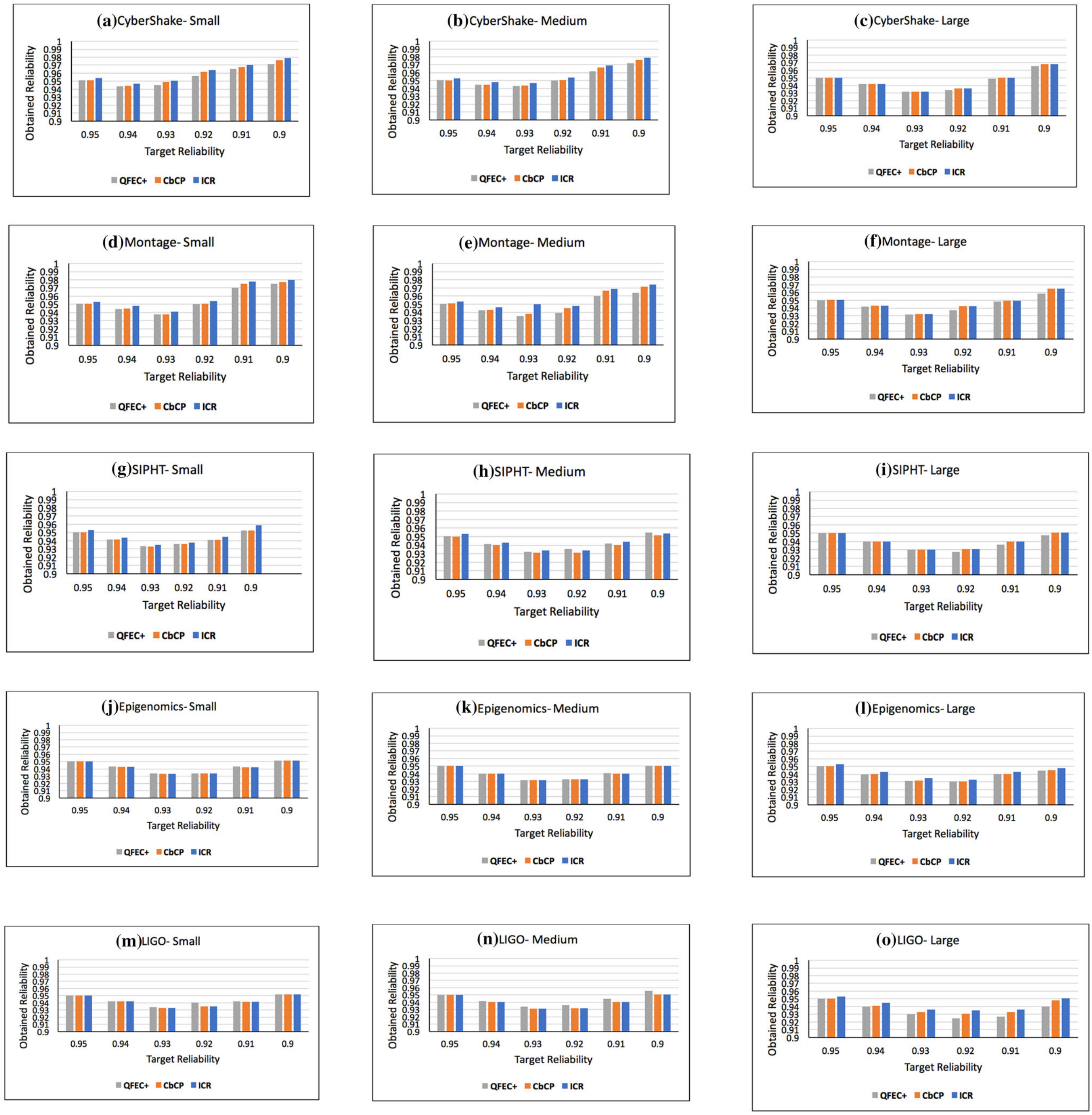


Fig. 6 Obtained Reliability of scheduling workflows with the ICR, CbCP and QFEC+ algorithms

the workflow execution cost improvement is almost zero. Because the rate of dependency between clusters is low in these workflows, their execution can be in parallel and so decrease the idle time slots in the resource. Therefore, the ICR and CbCP algorithms have almost the same normalized cost since the Fix Up algorithm cannot reduce their number of idle time slots.

In addition, Fig. 7 shows the results for a relaxed deadline and reliability. The ICR, CbCP, and QFEC+ algorithms have almost the same normalized cost for a relaxed deadline and reliability. Therefore, in cases in which the workflow deadline and reliability are not strict, the schedule map changes to reduce execution costs within the constraints of the specified deadline and reliability.

As demonstrated in Fig. 7, increasing the quantity of tasks from 25 to 100 lowers the normalized cost in Epigenomics while producing the same normalized cost in SIPHT and LIGO. However, in Montage and CyberShake, an increase in workflow tasks raises the normalized cost, because the structure of these workflows creates small

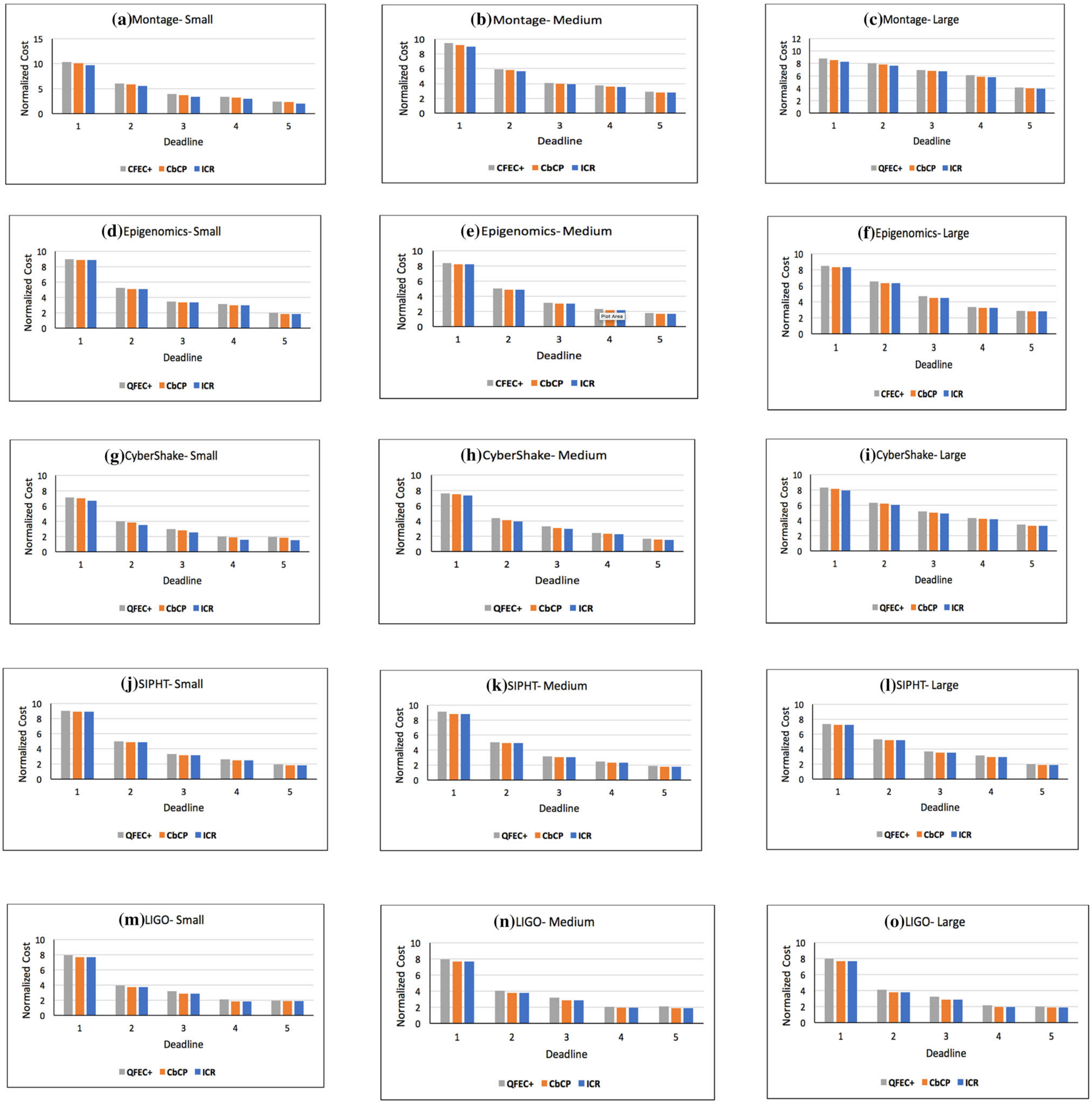


Fig. 7 Normalized Cost of scheduling workflows with the ICR, CbCP and QFEC+ algorithms

clusters. Therefore, the resource assignment algorithm must establish many resources, even though only a low number of their idle time slots are used.

6 Conclusions

Cloud computing enables users to gain their desired QoS (such as deadline and reliability) by paying an appropriate price. The present paper proposes a new algorithm, ICR, for workflow scheduling which minimizes the total execution cost while meeting a user-defined deadline and reliability.

When compared with CbCP and QFEC+ , the main advantages of ICR are its capability to start each task earlier in the leased resource, without imposing any extra cost, by shifting each task toward the beginning of the time interval and utilize the rest of the idle time slots in the leased resources to enhance the workflow execution reliability. The current study evaluates the ICR algorithm by simulating it with synthetic workflows based upon real scientific workflows of various structures and sizes. The outcomes show that ICR provides the best possible solution when the task graph satisfies a simple condition. Even if the condition is not met, this algorithm provides a satisfactory schedule close to the optimum solution. The shortest schedule length and resource usage are also of critical concern in high-performance computing systems. To address this, the proposed algorithm minimizes the schedule length via the Fix Up algorithm and considers resource usage by employing the Task Replication algorithm.

Future work shall focus on improving the proposed algorithm for utilization in scheduling multiple workflows whose requests are received at different rates.

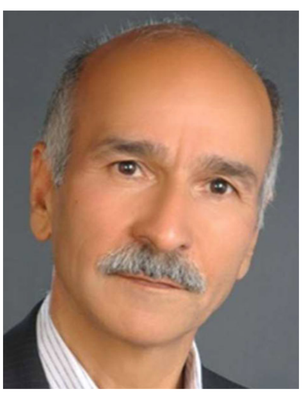
Acknowledgements The authors would like to express their gssratitude to the anonymous reviewers for their constructive comments which have helped to improve the quality of the paper.

References

1. Calheiros, R.N., Buyya, R., Member, S.: Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Trans. Parallel Distrib. Syst. **25**, 1787–1796 (2013)
2. Cai, Z., Li, X., Gupta, J.N.D.: Heuristics for provisioning services to workflows in XaaS clouds. IEEE Trans. Serv. Comput. **92**, 250–263 (2016)
3. Zhu, X., Wang, J., Guo, H., Zhu, D., Yang, L.T., Liu, L.: Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. IEEE Trans. Parallel Distrib. Syst. **27**(12), 3501–3517 (2016)
4. Zhou, A.: Cloud service reliability enhancement via virtual machine placement optimization. IEEE Trans. Serv. Comput. **10**(6), 902–913 (2016)
5. Zhao, L., Ren, Y., Sakurai, K.: Reliable workflow scheduling with less resource redundancy. Parallel Comput. **39**(10), 567–585 (2013)
6. Qiu, W., Zheng, Z., Wang, X., Yang, X., Lyu, M.R.: Reliability-based design optimization for cloud migration. IEEE Trans. Serv. Comput. **7**(2), 223–236 (2014)
7. Silic, M., Delac, G., Srbljic, S.: Prediction of atomic web services reliability for QoS-aware recommendation. IEEE Trans. Serv. Comput. **8**(3), 425–438 (2015)
8. Bajaj, R., Agrawal, D.P.: Improving scheduling of tasks in a heterogeneous environment. IEEE Trans. Parallel Distrib. Syst. **15**(2), 107–118 (2004)
9. Daoud, M.I., Kharma, N.: A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **68**(4), 399–409 (2008)
10. Wieczorek, M., Hoheisel, A., Prodan, R.: Towards a general model of the multi-criteria workflow scheduling on the grid. Futur. Gener. Comput. Syst. **25**, 237–256 (2009)
11. Yu, J., Kirley, M., Buyya, R.: Multi-objective planning for workflow execution on Grids. In: Proceedings on IEEE/ACM Int. Work. Grid Comput., pp. 10–17 (2007)
12. Dongarra, J.J., Jeannot, E., Saule, E., Shi, Z.: Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In: Proc. Ninet. Annu. ACM Symp. Parallel algorithms Archit.—SPAA '07, p. 280 (2007)
13. Swaminathan, S., Manimaran, G.: A reliability-aware value-based scheduler for dynamic multiprocessor real-time systems. In: Proceedings on Int. Parallel Distrib. Process. Symp. IPDPS 2002, no. December, p. 98 (2002)
14. Benoit A., Hakem, M., Robert, Y.: Fault tolerant scheduling of precedence task graphs on heterogeneous platforms. In: IPDPS Miami 2008—Proc. 22nd IEEE Int. Parallel Distrib. Process. Symp. Progr. CD-ROM, vol. 33, no. December 2007 (2008)
15. Benoit, A., Hakem, M., Robert, Y.: Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems. Parallel Comput. **35**(2), 83–108 (2009)
16. Girault, A., Kalla, H.: A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. IEEE Trans. Dependable Secur. Comput. **64**, 241–254 (2009)
17. Zheng, Q., Veeravalli, B.: On the design of communication-aware fault-tolerant scheduling algorithms for precedence constrained tasks in grid computing systems with dedicated communication devices. J. Parallel Distrib. Comput. **69**(3), 282–294 (2009)
18. Zheng, Q., Veeravalli, B., Tham, C.K.: On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs. IEEE Trans. Comput. **58**(3), 380–393 (2009)
19. Mousavi Nik, S.S., Naghibzadeh, M., Sedaghat, Y.: Cost-driven workflow scheduling on the cloud with deadline and reliability constraints. Computing **102**(2), 477–500 (2020)
20. Arabnejad, H., Barbosa, J.G.: A budget constrained scheduling algorithm for workflow applications. J. Grid Comput. **12**(4), 665–679 (2014)
21. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.D.: Scheduling workflows with budget constraints. In: Integr. Res. GRID Comput. CoreGRID Integr. Work. 2005 Sel. Pap., pp. 189–202 (2007)
22. Su, S., Li, J., Huang, Q., Huang, X., Shuang, K., Wang, J.: Cost-efficient task scheduling for executing large programs in the cloud. Parallel Comput. **39**(4–5), 177–188 (2013)
23. Szabo, C., Kroeger, T.: Evolving multi-objective strategies for task allocation of scientific workflows on public clouds. IEEE Congr Evol. Comput. CEC **2012**, 10–15 (2012)
24. Kianpisheh, S., Charkari, N.M.: A grid workflow Quality-of-Service estimation based on resource availability prediction. J. Supercomput. **67**(2), 496–527 (2014)
25. Xie, G., et al.: Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems. IEEE Trans. Serv. Comput. (2017)
26. He, Y., Shao, Z., Xiao, B., Zhuge, Q., Sha, E.: Reliability driven task scheduling for heterogeneous systems. Int. Conf. Parallel Distrub. Comput. Syst. (2003)
27. Qin, X., Jiang, H., Swanson, D.R.: An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. Parallel Process. 2002. In: Proceedings. Int. Conf., no. July, pp. 360–368 (2002)
28. Benoit, A., Hakem, M., Robert, Y.: Optimizing the latency of streaming applications under throughput and reliability constraint. In: Proc. Int. Conf. Parallel Process., pp. 325–332 (2009)
29. Zhao, L., Ren, Y., Sakurai, K.: A resource minimizing scheduling algorithm with ensuring the deadline and reliability in heterogeneous systems. In: Proc. - Int. Conf. Adv. Inf. Netw. Appl. AINA, pp. 275–282 (2011).
30. Xie, G., Zeng, G., Li, R., Member, S.: Quantitative fault-tolerance for reliable workflows on Heterogeneous IaaS clouds. IEEE Trans. Cloud Comput. (2017)
31. Naghibzadeh, M.: Modeling and scheduling hybrid workflows of tasks and task interaction graphs on the cloud. Futur. Gener. Comput. Syst. **65**, 33–45 (2016)

32. Benoit, A., Canon, L.C., Jeannot, E., Robert, Y.: Reliability of task graph schedules with transient and fail-stop failures: complexity and algorithms. J. Sched. **15**(5), 615–627 (2012)
33. Topcuoglu, H., Hariri, S.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**, 260–274 (2002)
34. Ranaweera, S., Agrawal, D.P.: A task duplication based scheduling algorithm for heterogeneous systems. Parallel Distrib. Process. Symp. 2000. IPDPS 2000. In: Proceedings. 14th Int., pp. 445–450 (2000)
35. Bharathi, S., Chervenak, A., Deelmn, E., Mehta, G., Su, M.H., Vahi, K.: Characterization of scientific workflows. In: 2008 3rd Work. Work. Support Large-Scale Sci. Work. 2008, no. June 2014, (2008)

Samaneh Sadat Mousavi Nik received the B.Sc. and M.Sc. degrees in Computer Engineering and Information Technology Engineering in 2007 and 2012, respectively, and started the Ph.D. degree in computer engineering at Ferdowsi University of Mashhad, in 2014. Her research interests focus on resource management and scheduling in the distributed systems, especially in Clouds. She has published several conference and journal papers.



Mahmoud Naghibzadeh has received the M.S. and Ph.D. degrees in Computer Science and Computer Engineering, respectively, from University of Southern California, USA. Now, he is a full professor at the Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran. In 1991 he was a visiting professor at University of California, Irvine, USA, and in 2003–2004 he was a visiting professor at Monash University, Australia. He is the director of Knowledge Engineering Research Group (KERG) laboratory and his research interests include the scheduling aspects of real-time systems, Grid, Cloud, Multiprocessors, Multicores, and GPGPUs. Besides, he is also interested in Bioinformatics computer algorithms, especially protein structures and protein-protein interactions. He has published numerous papers in international journals and conference proceedings as well as eight books in the field of Computer Science and Engineering. He has been the general chair of two international computer conferences as well as the technical chair of many others. Also, he is the reviewer of many journals and member of many computer societies and a senior member of the IEEE. Currently, he is the chief editor of the Computer and Knowledge Engineering (CKE) journal.

Yasser Sedaghat received the M.Sc. and Ph.D. degrees in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2006 and 2011, respectively. He is currently an Assistant Professor with the Department of Computer Engineering, Ferdowsi University of Mashhad (FUM), Mashhad, Iran. He has established and has been one of the chairs of the Dependable Distributed Embedded Systems Laboratory, FUM, since 2012. His current research interests include dependable embedded systems and networks, reliable software design, embedded operating systems, and field- programmable gate array based designs.