Computing (2020) 102:477–500
https://doi.org/10.1007/s00607-019-00740-5

Cost-driven workflow scheduling on the cloud with deadline and reliability constraints

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

Received: 29 January 2019 / Accepted: 2 July 2019 / Published online: 5 July 2019

Abstract
Clouds are becoming an effective platform for scientific workflow applications. In the meantime, Cloud computing structures are moving towards being more heterogeneous. In heterogeneous service-oriented systems, managing the reliability of resources (e.g., processors and communication networks) is widely identified as a critical issue due to processor and communication failures affecting user quality of service requirements. Therefore, these types of failures should be taken into account when scheduling algorithms. The present paper proposes a scheduling approach which includes four algorithms for minimizing the workflow execution cost while also meeting the user-specified deadline and reliability. To meet the application's requirements, the first algorithm partitions the workflow into several clusters based on a critical parent called CbCP. After that, the resource assignment algorithm, consisting of reliability and deadline distribution methods, satisfies the application's constraints. Experimental outcomes on various workflows, generated at different scales in real and random fashion, demonstrate that the proposed heuristics meet the deadline and reliability. This ensures the minimal cost when performing a similar quality of service as opposed to the performance of the state-of-the-art DRR and QFEC+ algorithms.

Keywords Cloud computing · Reliability · QoS-based scheduling · Clustering

Mathematics Subject Classification 68W01

✉ 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

1 Introduction

Scientific workflows are usually modeled as directed acyclic graphs (DAGs) whose nodes are regarded as tasks and whose directed edges constitute dependencies among tasks. Because of involving hundreds or thousands of tasks in a single workflow [1], a large-scale infrastructure, such as the public Cloud, can benefit the scientific workflow. For sharing, Cloud computing supplies an infrastructure for large scale and heterogeneous resources, such as computing and data resources. Numerous hardware and software capacities and configurations make resource management a critical issue in large scale heterogeneous systems. In order to perform scientific workflows over the Cloud, efficient scheduling algorithms must meet the demands of users or systems [1]. As workflow execution in the Cloud incurs financial cost to users, one of the Quality of Service (QoS) parameters is cost, in which the pricing model of the most prevalent commercial Clouds is based upon the quantity of time intervals utilized by users. In addition, the operating proficiency of processors in heterogeneous service-oriented architectures has been advanced to yield robust Cloud-based services, while processor and communication network failures [2] affect system reliability and user quality of service [3]. Therefore, reliability is another principal challenge in workflow scheduling [3–6]. Moreover, as a consequence of the availability of resources to applications dispensing dynamic scaling based on need, Cloud infrastructures are appropriate platforms for deadline-constrained workflow application execution. With all this variety, application developers, intent on providing their systems with the Cloud experience, face the challenge of making the best choices in regard to price, reliability, and deadline.

As scheduling is mapping tasks to the processors, the specific demands of QoS are met while task precedence constraints are considered. Since optimal task scheduling was verified to be NP-complete [7], numerous heuristic procedures have been recommended for homogeneous [8] and heterogeneous distributed systems [9, 10]. Thus, heuristics can be utilized to obtain sub-optimal schedules. The common task scheduling algorithms are categorized into a number of classes, such as list scheduling algorithms, cluster algorithms, and duplication-based algorithms. For instance, in regard to reducing the scheduling length, the HEFT list-scheduling algorithm [8] nominates tasks in accordance with the descending order of their upward rank and sends them to different processors. Therefore, the aim of the current paper is to obtain a schedule and gain the ability of satisfying a user-specified deadline in place of minimizing the execution time.

Furthermore, other items in the Cloud, such as workflow execution reliability as well as execution costs, are of reasonable importance. Reliability denotes the probability of successfully completing the execution of a schedule [3–6]. Improving reliability by redundancy in space and time has been proposed by numerous algorithms [12, 13]. Redundancy in space is one of the widespread mechanisms for providing fault tolerance; for example, [2] applies an active replication scheme to maximize execution reliability. This replication plan concurrently carries out the replication of each task on various processors and the task succeeds if at least

one of the processors does not fail [11–13]. Besides, more replicas imply more resource consumption and higher economic cost. Therefore, the aim of the current paper is to obtain a schedule with the property of satisfying a user-specified reliability in place of maximizing the execution reliability.

Another processor failure mitigation technique is the backup/restart procedure which deals with rescheduling the task on a backup processor so as to ensure progress [14, 15]. An enhanced form of the backup/restart procedure employed in the case of failure is the checkpoint/restart scheme, in which the task is restarted from the most recent checkpoint instead of from the exact beginning. By relying on the redundancy in time, missed deadlines may occur in backup/restart, as well as in checkpoint/restart schemes. As satisfying the user defined deadline is an essential issue in our work, this procedure cannot be employed in current study.

Current research has begun to study the fault-tolerant scheduling algorithm based on exploring minimum numbers of replicas to satisfy the reliability requirement of the workflow. For example, Zhao et al. [4], Xie et al. [16] proposed a resource redundancy minimizing algorithm (DRR) with deadline and reliability requirements and quantitative fault-tolerant scheduling algorithm QFEC+ with minimum execution costs, respectively. In both procedures, firstly, probability of failure for a task on a processor is calculated. Then, by respecting each task's sub-reliability requirement, different tasks are replicated a number of times and are sent to corresponding processors. However, a major limitation of DRR and QFEC+ is ignoring the workflow structure, especially when there are high interdependencies among tasks.

Since processor and communication failures affect user quality of service requirements, it is essential to take into account the processors' probability of failure as well as related network resources when scheduling algorithms.

To tackle this issue and also meet the reliability and deadline requirements of users at a minimum cost, the current study proposes a new reliable scheduling approach: CbCP. The first algorithm partitions the workflow into several clusters based on the critical parent (CbCP). Reliability and deadline distribution algorithms are then introduced to meet the application's reliability and deadline requirements while the overall cost of execution is minimized.

In accordance with these challenges, the contributions of the present work are summarized as follows:

- It is found that the key point for increasing reliability is considering the workflow structure, especially when there are high interdependencies among tasks. Reducing the number of messages transferred from parent tasks to children tasks can enhance the obtained workflow reliability. Therefore, clustering based on the critical parent (CbCP) algorithm is proposed to partition the workflow into clusters that are able to be performed in parallel or serial.
- The reliability distribution algorithm is introduced to satisfy the reliability demand of applications in two stages. The first stage includes gaining the lower bound on the sub-reliability demand of each cluster and the second stage is repeatedly choosing the accessible processor with the minimum cost values till its sub-reliability demand is met.

- The deadline distribution algorithm is presented to detect the cheapest schedule that can execute each workflow task prior to its latest finish time. As a workflow includes several tasks, each task is related to a sub-deadline in accordance with the overall deadline. In this algorithm, the sub-deadlines of tasks are assessed by a traversal of the workflow graph in reverse topological order.
- Experimental outcomes on various workflows, generated at different scales in real and random fashion, validate that the proposed approach can incur the lowest execution cost when compared with the state-of-the-art DRR and QFEC+ algorithms.

In the remaining parts of the current paper, Sect. 2 reviews the related works, while Sect. 3 explains the system modeling, problem definitions, and evaluation metrics. Section 4 discusses the features of the proposed approach and the complexity and illustrations of the examples. Section 5 provides the simulation outcomes while Sect. 6 presents the conclusion and future work.

2 Related work

Workflow scheduling algorithms are categorized into two principal classes: QoS constrained and QoS optimization [17]. The QoS constrained algorithms attempt to satisfy user specified QoS requirements while optimizing some QoS specifications. For example, some papers have explored limiting the budget while minimizing makespan. Scheduling workflows which minimizes the makespan is established by Sakellariou et al. [18]. The schedule is completed if the cost is within the budget or tasks are remapped to cheaper processors, which can thus meet budget constraints. The QoS optimization algorithm attempts to optimize all QoS specifications. In this area, some studies have been conducted to yield a balance between time and cost [19, 20]. It is assumed that the processors are accessible at any desired time. However, in reality, processor and network failure is unavoidable. Resources may become inaccessible due to reasons such as link failure, power variation, and software/hardware failures [21]. Therefore, to decrease workflow execution failure, it is necessary to consider the reliability of well-organized workflow scheduling. According to the common exponential distribution assumption in the field of reliability [22], the occurrence of failure for each processor follows the Poisson distribution with a failure rate (λ), which is a positive real number identical to the expected number of occurrences of failure in unit time t. Accordingly, reliability during the interval of time t is $e^{-\lambda t}$. To satisfy a specified time constraint while bringing about maximum system reliability, the Minimum Cost Match Schedule (MCMS) and Progressive Reliability Maximization Schedule (PRMS) were developed in [25]. Evidently, the reliability obtained by these works is limited and special schemas, such as active replication, are mandatory. The primary and backup scheduling algorithm can tolerate one failure in the system. Principal popular procedures include the efficient Fault-tolerant Reliability Cost Driven (eFRCD) [23], efficient Fault-tolerant Reliability Driven (eFRD) [12], and Minimum Completion Time with Less Replication Cost (MCT-LRC) procedures [15]. All of these procedures assume that no more than one failure occurs

at any time. In order to assure system reliability, [11] utilizes $\varepsilon+1$ replicas for each task to design the Fault-tolerant Scheduling Algorithm (FTSA). In addition, Benoit et al. in [24] present another scheduling algorithm to minimize the schedule length under the throughput as well as reliability constraints for parallel application on heterogeneous systems in the active replication procedure. The basic issue in [11, 24] is the need for ε backups for each task with excessive redundancy in order to meet application reliability constraints. This causes great redundancy in the resources and so adversely affects system performance and produces high resource costs. Current research has begun to study active redundancy for each task procedure so as to meet application reliability constraints [4, 16, 25]. In an active redundancy scheme based on active replication, each task will have its own number of replicas which leads to lower resource costs than those of fixed ε backups for different tasks [25]. Zhao et al. [4, 25] and Xie et al. [16] propose the fault-tolerant scheduling algorithms of MaxRe, reliability requirements to minimize resource redundancy (RR) and quantitative fault-tolerant scheduling algorithm QFEC+ with minimum execution costs, respectively. All of these procedures combine reliability analysis into the active replication and utilize a dynamic quantity of backups for various tasks by taking into account each task's sub-reliability demand. One of MaxRe, RR and QFEC+ limitations is in computing the sub-reliability demands of tasks. In [4], the authors further suggest the DRR algorithm, which applies the deadline requirement of a parallel application in RR. Moreover, neither of DRR nor QFEC+ consider the workflow structure which is an important issue when there are high interdependencies among tasks. Thus, the main differences between our work with [4, 16], is the methods of allocation and calculating the sub-reliability requirement of each task.

The current paper's interest is to meet the reliability and deadline requirement. However, some of the above-mentioned algorithms do not take into account the communication between tasks and link failures in a network.

3 System model and problem definition

This section explains workflow and Cloud modeling, followed by a formal outline of the problem.

3.1 Application and cloud models

The workflow model is one of the most successful patterns for programming scientific applications on distributed infrastructures, such as the grid and Cloud. Bags of tasks is another successful application covered by this model [26]. The work on the current scheduling algorithm concentrates on scheduling scientific workflows in the Cloud.

Since the tasks in a workflow are optionally interconnected, the application model utilized for a scientific workflow is a directed acyclic graph (DAG) [27]. Let $G_\tau = (V, E, ET, CM)$ be the graph corresponding to a workflow in which

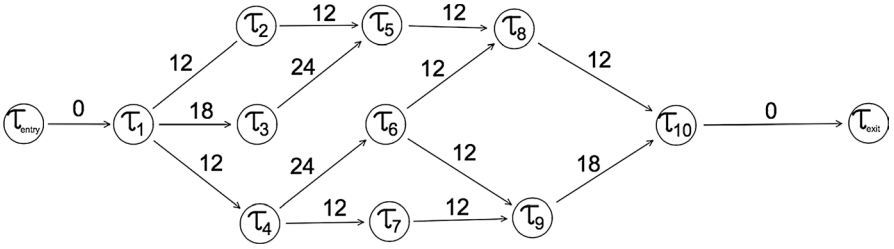

Fig. 1 A sample workflow

Table 1 Available resources for the workflow of Fig. 1

RS	*ET*										*PR*	Λ
	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}		
rs_1	12	30	18	24	18	24	30	18	30	12	5$	0.00010
rs_2	30	72	30	36	48	48	48	36	48	30	2$	0.00015
rs_3	48	96	54	60	66	66	66	48	84	48	1$	0.00018

$V = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is the set of tasks. E is the set of edges describing the precedence relationships among tasks. Edge $(e_i, e_k) \in E$ indicates that τ_i is a parent of τ_k. Since the proposed algorithm needs a single entry and a single exit task, two dummy entry and exit tasks, represented by τ_{entry} and τ_{exit} and with zero processing time and zero communication, are assumed at the start and at the end of the workflow, respectively. ET is an $n \times m$ matrix in which $exe_{i,j}$ indicates that the execution time of τ_i occurs on resource rs_j. Since Cloud is heterogeneous in nature, the computation time of tasks differs from processor to processor. CM is an $n \times n$ matrix where $cm_{i,k}$ indicates the size of data transferred from τ_i to τ_k. The amount of data communication among tasks is predetermined and known in advance. In a workflow, a task can begin execution when the executions of all of its parents have been completed and its essential data has been transferred.

For scheduling the algorithm, the present work models the Cloud resource by $G_{rs} = (RS, \Lambda, PR)$, where RS denotes the finite set of m heterogeneous processors: $RS = \{rs_1, rs_2, \ldots, rs_m\}$. $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ represents the failure rate of rs_j which is a positive real number identical to the expected number of occurrences of failure in unit time t and $PR = \{pr_1, pr_2, \ldots, pr_m\}$ indicates the price of each resource per time unit.

Figure 1 presents a sample workflow consisting of ten tasks from τ_1 to τ_{10}. Since the proposed algorithm needs a single entry and a single exit task, two dummy tasks, τ_{entry} and τ_{exit}, are considered. In Fig. 1, each weighted edge shows both the estimated data transfer time and the precedence constraint between the corresponding tasks. Three different possible resources are assumed for each task τ_i, i.e., rs_1, rs_2, and rs_3, which can execute the task with a different QoS. Table 1, indicates the execution times of tasks on different resources (ET), the price of each resource per time unit (PR) and failure rate of each resource (Λ), respectively. As it is seen, for the processing of τ_5 to begin, all the data needed from τ_2 and τ_3 must be received. In

Table 2 Important notations in this study

Notation	Definition
$exe_{i,j}$	Execution time of task τ_i on resource rs_j
$cm_{i,k}$	Communication time between the tasks τ_i and τ_k
λ_j	Constant failure rate of resource rs_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 resource rs_j
$CMR_{k,i}$	Communication reliability between parent and its child tasks.
$CR_{i,j}$	Computation reliability of task τ_i on the resource rs_j
$CR_{cl_x j}$	Computation reliability of cluster x (cl_x) on the resource rs_j
CP_i	Critical parent of task τ_i
$Rel_{schedul}$	Reliability 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 resource rs_j
Rel_j^i	reliability attained by scheduling a task τ_i on resource rs_j

this example, the makespan of the workflow will be the finish time of the end task, which is τ_{10}.

3.2 Problem definition

As discussed in Sect. 1, it is not possible for any workflow execution to be 100% reliable. However, if the system can satisfy the workflow's reliability requirement, then the workflow execution is considered reliable. The problem addressed by the present study may be formally described as follows. The input of the scheduling system includes the given workflow G_τ, deadline D, and reliability R. The problem is to find a map of tasks for processors to generate a concrete workflow, such that the *Cost* of workflow execution is minimized while the makespan satisfies the deadline and the obtained reliability of the workflow, $Rel_{schedul}$, meets the workflow's reliability requirement. Equation (1) defines the formal description of the program and Table 2 provides the list of important notations and their definitions that are used in this study.

$$\begin{aligned} &Min\ Cost = \sum_{x=1}^{y} \left\lceil \frac{TET_{cl_x j}}{lenght\ of\ interval} \right\rceil \times pr_j \\ &M = FT_{\tau_{exit}} - ST_{\tau_{entry}} \leq D \\ &Rel_{schedul} \geq R \end{aligned} \tag{1}$$

where *Cost* is the cost that the user should pay for processor and network consumption to execute the workflow. As mentioned previously, our paper is based on clustering algorithm. In other words, according to critical parent concept, the workflow is partitioned into some clusters in which y is the number of generated clusters. $TET_{cl_x j}$ be the total execution time of cluster x (cl_x) on resource rs_j. $Rel_{schedul}$ is the

reliability of workflow execution, which is defined as the probability of the workflow execution being successfully completed. M is the workflow makespan, which is defined as the time it takes to complete the execution of all tasks. Finally, the start time and finish time of each task are also defined during scheduling.

To model the cost of resource usage, one of the available pricing models employed by most commercial infrastructure as a service (IaaS) Cloud service providers is based on a pay-as-you-go approach. In this model, the users are charged according to the number of time intervals they have used the resource, even if the last time interval was not completely used.

Transient failures (also called random hardware failures) and permanent failures are the two main failure categories for modeling reliability. Once a permanent failure occurs, the processor can only be restored by replacement. On the other hand, in the case of a transient failure, which is the most probable occurrence, failure appears for a short time and then disappears without damage to the processors [22]. Therefore, the current paper mainly takes transient failures into account for its research. In general, in the case of a transient failure, a task in a DAG-based application follows the Poisson distribution with failure rate (λ) [4, 13, 28]. Therefore, Eq. (2) denotes the reliability of an event in unit time t:

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

In addition, the current paper considers heterogeneous systems that consist of various hardware and software with different configurations or capacities. Thus, the Mean Time Between Failure (MTBF) of each processor differs from one to another. Similar to [28], the present study considers the assumption that no failure happens during the processor's idle times.

4 The proposed scheduling approach

The motivation behind the present research is the QoS constrained static scheduling of scientific workflows on the IaaS Cloud platform. The data communications and precedence constraints among workflow tasks make optimal scheduling of the workflow more complicated and so more difficult to attain. The issue of reliability-aware design adds some complexity to this scheduling problem, because reliability depends on Communication Reliability, *CMR*, and Computation Reliability, *CR*.

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

$CMR_{k,i}$ is the probability that the message created by the parent tasks, for example τ_k, can be successfully moved to the processors where their child tasks, for example τ_i, are located. $CR_{i,j}$ is the probability that τ_i is successfully completed on rs_j.

The key point for increasing reliability is to consider the workflow structure. Based on Eq. (3), reducing the number of messages transferred from parent tasks to children tasks can enhance achieving the reliability of the workflow. Therefore, as the proposed algorithm is based on clustering, different scheduling algorithms have

been studied that address the concept of workflow clustering [29, 30]. The present research provides an algorithm which finds clusters in accordance with the critical parent. In order to select the critical parent in each step, a scoring function has been employed.

This section first discusses the main ideas of the proposed approach and then provides some basic definitions. Afterwards, the CbCP scheduling approach and computation of its time complexity is elaborated upon. Finally, there is a demonstration of its operation by an illustrative example.

4.1 Main ideas

The proposed approach consists of four main algorithms: clustering, reliability distribution, deadline distribution, and resource assignment. The clustering algorithm is based on a critical parent, which means that, for each τ_i, a critical parent, CP_i, is assigned by Eq. (4). This equation designates the assigning task as well as its critical parent to the identical processor. $pred_i$ is the set of immediate predecessor tasks of τ_i. The earliest time when τ_k can finish its computation is defined as its Earliest Finish Time, EFT_k, which will be explained in the next sub-section.

$$CP_i = k \in pred_i \; s.t \; EFT_k + cm_{k,i} > EFT_l + cm_{l,i} \; where \; l \in pred_i \; and \; k \neq l \tag{4}$$

R represents the reliability demand. In the reliability distribution algorithm, the sub-reliability requirement for the clusters is continuously calculated based upon the actual reliability obtained by previous assignments so as to ensure that the overall reliability requirement is met. Finally, the resource assignment chooses the cheapest service for each cluster while satisfying its sub-reliability and sub-deadline. 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. The following sub-sections discuss each algorithm in detail.

4.2 Basic definitions

In the proposed CbCP scheduling approach, the aim is to observe the priority and critical parent of all tasks. In order to determine these, it is necessary to compute some parameters of each workflow task before scheduling the workflow.

The length of the longest path, from the task to τ_{exit}, is the rank of a task indicated by $Rank_i$. When the length of the path is calculated, the computation times are considered, but the communication times are ignored.

$$\begin{aligned} &Rank_i = \overline{exe}_{i,j}, && if \; succ_i = \varphi, \; \tau_i \in V, \; rs_j \in RS \\ &Rank_i = max_{\tau_k \in succ_i}\left(Rank_k + \overline{exe}_{i,j}\right), && Otherwise \end{aligned} \tag{5}$$

The rank is repeatedly calculated by passing over the task upwardly starting from τ_{exit}. In Eq. (5), $\overline{exe}_{i,j}$ is the average computation time of task τ_i and $succ_i$ is the set of immediate successor tasks of τ_i.

The earliest start time, EST_i of each unscheduled task τ_i is defined as the earliest time when τ_i can begin its computation. Since Cloud is heterogeneous in nature and the computation time of tasks differs from processor to processor, it is impossible to compute the exact EST_i. Therefore, the execution and data transmission time for each unscheduled task should be estimated. The minimum execution and data transmission time are selected among other potential approximation alternatives, e.g., the average or the median. Therefore, the Earliest Start Time, EST_i, is defined as follows:

$$\begin{aligned} &EST_i = 0, \quad \textit{for the entry node} \\ &EST_i = max_{k \in pred_i} \left(EFT_{k \text{ where } \tau_i \& \tau_k \in \text{identical rs}} \parallel EFT_{k \text{ where } \tau_i \text{ and } \tau_k \notin \text{identical rs}} + cm_{k,i} \right) \end{aligned} \tag{6}$$

Moreover, for each unscheduled task τ_i, the earliest time when τ_i can finish its computation is defined as its Earliest Finish Time, EFT_i. Once again, it is impossible to precisely compute EFT_i and it should be computed in accordance with 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, which is the latest time when τ_i can finish its computation, may be defined as:

$$\begin{aligned} &LFT_i = Deadline, \quad \textit{For the exit node} \\ &LFT_i = \min_{k \in succ_i} \left(LST_{k \text{ where } \tau_i \text{ and } \tau_k \in \text{identical rs}} \parallel LST_{k \text{ where } \tau_i \text{ and } \tau_k \notin \text{identical rs}} - cm_{i,k} \right) \end{aligned} \tag{8}$$

Finally, the Latest Start Time,LST_i, which is the latest time when τ_i can begin its computation, is expressed in (9):

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

4.3 Clustering based on critical parent

Algorithm 1 shows the pseudocode of the overall CbCP algorithm for scheduling a workflow. Two dummy nodes, τ_{entry} and τ_{exit}, are added to the task graph in Line 3, even if the task graph previously just had one entry or exit node. The task graph is traversed to compute the desired parameters in Lines 4–9. After this, a sub-deadline is allocated to nodes τ_{entry} and τ_{exit}(Line 10). Accordingly, the user's deadline is employed to set the sub-deadline of τ_{exit}. This causes the parents of τ_{exit}, i.e., the actual exit node of the workflow, to be performed prior to the deadline. The last two lines are the major part of the algorithm. For the input node, the clustering procedure is called in Line 11. This method partitions the workflow into some clusters according to critical parent concept. At the end, in Line 12, the process of resource assignment is called to choose the cheapest processor for each cluster in accordance with the sub-deadline and sub-reliability for each task.

Algorithm 1. The CbCP Scheduling Algorithm

1. **procedure** ScheduleWorkflow ($G_\tau(V, E, ET, CM)$, D , R)
2. determine available computation processors
3. add τ_{entry}, τ_{exit} and their corresponding edges to G
4. compute $Rank_i$ for each task according to Eq. 5, starting from the τ_{exit}
5. compute EST_i for each task according to Eq. 6
6. compute EFT_i for each task according to Eq. 7
7. compute LFT_i for each task according to Eq. 8
8. compute $LST_i$for each task according to Eq. 9
9. compute $CP_i$for each task in G_τ according to Eq. 4
10. $EST(\tau_{entry}) \leftarrow 0$, $LFT(\tau_{exit}) \leftarrow$ D
11. call clustering (G_τ)
12. call Resource Assignment (G_τ)
13. **end procedure**

4.3.1 Clustering algorithm

Based upon a clustering heuristic, the reliability and deadline distribution algorithms are the main contributions in the current paper. An algorithm in this group maps the tasks which communicate heavily on the same cluster [31].Although the proposed clustering algorithm is based on the same heuristic, it utilizes the CP for selecting the task for each cluster.

4.3.1.1 Task priority Several approaches exist for computing the task priority, for example: the upward rank, the upward rank + downward rank [32], and schedule pressure [33]. Generally, the upward rank and the upward rank + downward rank are the two most common approaches for prioritizing tasks. In the proposed CbCP algorithm, tasks are ranked by their priorities which are based on the upward rank.

The priority of task τ_i is the length of the longest path from τ_i to τ_{exit}. When computing the priority or the length of the path, the communication times are ignored and only the computation times are taken into account according to Eq. (5). The priority of τ_{entry} is the sum of computation costs through the longest path. This schedule length can never be lower than the priority of the τ_{entry} of DAG.

Algorithm 2. Clustering Generation Algorithm

1. ***procedure*** *clustering(G_τ)*
2. *sort the tasks in a scheduling list by ascending order of rank value*
3. ***while*** *(there exists an unassigned task) do*
4. *Select the first task τ_i from the list for clustering*
5. ***if*** *the τ_i is critical parent of any successor tasks* ***then***
6. *add τ_i to the proper cluster*
7. *remove τ_i from the list*
8. ***else***
9. *create new cluster*
10. *add τ_i to this new cluster*
11. *remove τ_i from the list*
12. ***end if***
13. ***end while***
14. ***end procedure***

4.3.1.2 Clustering generation algorithm Algorithm 2 demonstrates the pseudocode for clustering. This algorithm creates the task clusters based upon the parameters calculated in Algorithm 1 and the rank list. Algorithm 2 arranges the nodes of the task graph by the smallest rank first order in the list (Line 2). The creation of a cluster is initiated from the list's first task, which has not yet been allocated to a cluster. The allocation is conducted by following the selected task, which is the critical parent of any allocated successor tasks. Then the current task is added to the proper cluster (Lines 4–7). Otherwise, a new cluster is generated for this task (Lines 9–11). The generation of the cluster is completed when there is no longer any unassigned task in the list.

4.3.2 Resource assignment algorithm

In order to schedule cluster x (cl_x) to a processor with a minimum cost while also respecting both the reliability and deadline constraints, the current work proposes a new deadline and reliability distribution. Algorithm 3 describes the heuristic algorithm of resource assignment.

Let D and R be the overall deadline and reliability requirements of an application respectively. Since a workflow contains of a number of tasks, each task is related to a sub-deadline based on the overall deadline. Therefore, the sub-deadlines of exit tasks are identical to D and the sub-deadlines of the remaining tasks are computed according to a traversal of the workflow graph in reverse topological order. Moreover, the reliability requirement of the application is transferred to the sub-reliability requirement of each cluster. The details are explained as follows.

Algorithm 3. Resource Assignment Algorithm

```
1.  procedure Resource Assignment(G_τ)
2.  sort the tasks in a scheduling list by ascending order of rank value
3.      while (there exists an unscheduled task) do
4.          τ_i ← delete first task from the list
5.          if τ_i is the first task of the cluster then
6.              Compute the Rel_min^{cl_x} using Eq. 10
7.              if Rel_min^{cl_x} ≥ 1 then
8.                  Reject the scheduling
9.              end if
10.             Launch a new instance rs_j of cheapest service RS which can finish each task of cluster before its
    LFT_i and satisfied Rel_min^{cl_x}
11.         else
12.             rs_j ← add to the cheapest applicable existing instance for its cluster
13.         end if
14.     compute AST_i according to Eq. 15
15.     update LFT for all unassigned predecessors of τ_i according to Eq. 17
16.     end while
17. end procedure
```

4.3.2.1 Reliability distribution algorithm In order to schedule cluster x (cl_x) to a processor so as to minimize costs by meeting the reliability constraint, the current work computes the sub-reliability requirement of each cluster. Let R be the reliability

requirement and y be the number of generated clusters. Thus, the minimum sub-reliability requirement for cl_x ($Rel_{min}^{cl_x}$) with the highest priority is computed as:

$$\begin{aligned} Rel_{min}^{cl_1} &= \frac{R}{1 \times \prod_{i=x+1}^{y} Rel_{max}^{cl_{i,j}}}, \quad x = 1 \\ Rel_{min}^{cl_x} &= \frac{R}{\prod_{i=1}^{x-1} Rel_{Actual}^{cl_i,j} \times \prod_{i=x+1}^{y} Rel_{max}^{cl_{i,j}}}, \quad 2 \le x \le y,\ rs_j \in RS \end{aligned} \tag{10}$$

The details are explained as follows.

At first, the algorithm provided the minimum reliability requirement of the current cluster prior to its consideration for assignment according to Eq. (10), where $Rel_{Actual}^{cl_i,j}$ is the probability that cl_x is successfully executed on processor rs_j and $Rel_{max}^{cl_{i,j}}$ is the probability that cl_x is successfully executed on processor rs_j with the highest computation reliability processor RS.

Then, for choosing the proper processor, one possible method is iteratively selecting the available processor with the minimum cost value for the current cluster. After that, the actual reliability value of the current cluster is computed based on Eq. (12). Until the sub-reliability requirement is met, it is assumed that the reliability attained by scheduling task τ_i on rs_j is Rel_j^i.

$$\begin{aligned} Rel_j^i &= CR_{i,j}, \quad \textit{if there is no communication with other clusters} \\ Rel_j^i &= CR_{i,j} \times CMR_{k,i}, \quad \tau_i \in V,\ (e_k, e_i) \in E,\ rs_j \in RS \\ CR_{i,j} &= e^{-\lambda_j exe_{i,j}} \end{aligned} \tag{11}$$

$$Rel_{Actual}^{cl_{x,j}} = \prod_{\forall \tau_i \in cl_x} Rel_j^i, \quad rs_j \in RS \tag{12}$$

Since the proposed method is based on clustering and all the tasks in one cluster are assigned to the same processor, it is possible to assume that each cluster resembles one big task. Therefore, to decide which processors can satisfy the reliability requirement of the current cluster, the proposed method iteratively selects the available processor with the minimum cost value. Then, the total execution time, $TET_{cl_x,j}$, and the actual reliability value, $Rel_{Actual}^{cl_{x,j}}$, of the current cluster for each processor are computed based on Eqs. (13) and (14) until the sub-reliability requirement is met.

$$TET_{cl_x,j} = exe_{i,j} + \sum_{\tau_k \in rs_j} exe_{k,j}, \quad rs_j \in RS \tag{13}$$

$$\begin{aligned} Rel_{Actual}^{cl_{x,j}} &= CR_{cl_x,j} \times \prod_{\forall \tau_i \in cl_x} CMR_{k,i}, \quad (e_k, e_i) \in E,\ \tau_k \notin cl_x \\ CR_{cl_x,j} &= e^{-\lambda_j TET_{cl_x,j}} \end{aligned} \tag{14}$$

$TET_{cl_x,j}$ is the total execution time of cl_x on rs_j, where $exe_{i,j}$ is the execution time of τ_i on rs_j. $\sum_{\tau_k \in rs_j} exe_{k,j}$ is the sum of execution time of the previous tasks that have already been scheduled on rs_j. Therefore, $Rel^{cl_{x,j}}_{Actual}$ is the probability that cl_x is successfully executed on processor rs_j during the $TET_{cl_x,j}$ period.

Clearly, since Cloud is a heterogeneous environment, the computation time of clusters and computation reliability varies from one processor to another.

According to Eq. (10), for the rest of the clusters, their sub-reliability requirements are continuously calculated based on the actual reliability achieved by previous allocations, as well as presuppose reliability achieved by assigning unallocated clusters to the processor with maximum reliability.

Actual reliability depends on reliability provided by computation reliability (CR) and communication reliability (CMR), which are computed from tables of reliability given by the Cloud provider. For communication reliability, the location of parent tasks as well as child tasks is considered.

Compared with the DRR and QFEC+ algorithms, the principal enhancement of the presented CbCP is that it recomputes the sub-reliability requirement of each cluster, in regard to not only its previous allocations $Rel^{cl_{i,j}}_{Actual}$, but also succeeding pre-assignments $Rel^{cl_{i,j}}_{max}$.

4.3.2.2 Deadline distribution algorithm Finding the cheapest schedule that can complete each task of the workflow prior to its latest finish time is the goal of the present study's approach. Consequently, this method is utilized for assigning sub-deadlines to tasks on the list. According to Algorithm 3, it starts from the first task on the rank list and moves forward to the last task. As stated by Eq. (16), the sub-deadline of τ_{exit} is set to the workflow deadline. That is, the actual exit nodes of the workflow, i.e., the parents of τ_{exit}, must be completed before the user deadline.

At first, for each task, resource assignment to its cluster requires to check if no resource is allocated previously to that cluster. The algorithm examines the processors for a task, from the slowest to the fastest one. Afterwards, the admissible assignment processor for the current task (Line 10) is selected, which satisfies the sub-deadline and sub-reliability. Then, according to Eq. (15), the actual start time of this task may be calculated. It should be noted that, when an *AST* for the current task is assigned, the *LFT* of its unassigned predecessors may change [Eq. (17)]. For this reason, the algorithm updates this value for them.

$$AST_i = LFT_i - exe_{i,j} \tag{15}$$

$$LFT_i = Deadline, \quad For\ the\ exit\ node \tag{16}$$

$$LFT_k = \min\Big(AST_{i \in succ_k \text{ where } \tau_i \text{ and } \tau_k \in \text{identical rs}} \parallel AST_{i \in succ_k \text{ where } \tau_i \text{ and } \tau_k \notin \text{identical rs}} - cm_{k,i}\Big), \quad unassigned\ pred_i \tag{17}$$

On the other hand, if the cluster is previously allocated to any processor, then the algorithm will assign the task to that processor. If the assignment is admissible, the algorithm calculates the *AST* for the current task and the *LFT* of its unassigned

predecessors. Nonetheless, in the case of an inadmissible assignment, the algorithm backtracks to the foregoing task on the cluster in order to make another attempt.

4.4 Time complexity

In order to compute the time complexity of the proposed algorithm, it is assumed that the schedule workflow receives workflow $G_\tau = (V, E)$ as an input with n tasks and e edges. In addition, the maximum number of resource types for each task is assumed to be m. Since G_τ is a directed acyclic graph, the maximum number of assumed edges is $O(V^2)$. Therefore, the time complexity of the algorithm's main parts is computed as follows:

At first, the algorithm traverses each task of the task graph and computes the start times and completion times. At each node, the incoming and outgoing edges are examined and, in the worst-case, all the DAG edges must be examined. Thus, the worst-case complexity of these steps is $O(|E|)$, where $|E|$ is the number of edges.

Since the array queue needs to be generated, this can be done at time $O(|V|log|V|)$, which is the time for sorting the DAG nodes in ascending order of priority [34].

For the resource assignment algorithm, all resources for each cluster should be tried in order to find the cheapest one that respects both the sub-reliability and sub-deadline constraints. In each attempt, the actual start time of the task on that resource ought to be computed. To achieve this, all parent tasks and their edges must be considered. In the worst case, in which a node has n-1 unassigned predecessors, the time complexity of updating LFT for all nodes will be $O(|V|)$.

Thus, the overall time complexity of the CbCP algorithm is O(|V| + |E| + |V| log|V|). For a dense graph, the number of edges is proportional to O($|V|^2$). Thus, the worst-case complexity of the CbCP algorithm is O($|V|^2$).

4.5 An illustrative example

In order to show how the algorithm works, Fig. 1 traces its operation on a sample graph. 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 possible resources for each task τ_i, i.e., rs_1, rs_2, and rs_3, which can execute the task with a different QoS. Table 1 indicates the execution times of tasks on different resources (ET), the price of each resource per time unit (PR) and failure rate of each resource (Λ), respectively. Furthermore, all processors are assumed to be completely available and able to be provided at any desired time. As seen, for each faster resource, costs and reliability are greater than those of a slower one. Since any amount of communication inside the Cloud is free [29], the Cloud's data transfer cost of the workflow between resources is assumed to be zero.

In Fig. 1, each weighted edge indicates both the estimated data transfer time and precedence constraint between the corresponding tasks. Finally, the overall deadline and reliability of the workflow are 300 and 0.94, respectively.

Table 3 The Values of Rank, Critical Parent (CP), EST and EFT for Each task of the Workflow of Fig. 1

Task	$Rank_i$	CP_i	EST_i	EFT_i
τ_1	204	–	0	12
τ_2	174	–	24	54
τ_3	142	τ_1	30	48
τ_4	172	–	24	48
τ_5	108	τ_3	72	90
τ_6	130	τ_4	72	96
τ_7	132	–	60	90
τ_8	64	τ_6	108	126
τ_9	84	τ_7	108	138
τ_{10}	30	τ_9	156	168

When the CbCP scheduling algorithm (Algorithm 1) is called for the sample workflow (Fig. 1), it first computes the rank for each workflow task according to Eq. (5). Then, Earliest Start Time and Earliest Finish Time are computed by assigning the tasks to their fastest processor. To determine the CP of each task, the nodes of the task graph are sorted according to the smallest rank order in the list. The determination of a CP is initiated from the first task in the list. For instance, τ_{10} has two predecessors, in which τ_9 is its CP. The process is completed when there is no longer an unchecked task in the list. Table 3 provides the initial value of these parameters.

The next steps are to call the main procedures of the algorithm: the clustering generation algorithm (CGA) and resource assignment, which shall be discussed.

The generation of a cluster is initiated from the first task in the list which has not yet been assigned to a cluster. Based on CGA, the tasks in a scheduling list are sorted by the ascending order of rank value. Therefore, the array list for this DAG is:$List = \{\tau_{10}, \tau_8, \tau_9, \tau_5, \tau_6, \tau_7, \tau_3, \tau_4, \tau_2, \tau_1\}$. The first unassigned task in the list is τ_{10}. Because it is not a critical parent of any successor task, a new cluster is generated for this task. The condition of the next task, τ_8, is the same as that of τ_{10}. However, τ_9 is CP_{10} and so τ_9 is chosen for allocation to the same cluster as that of τ_{10}. The rest of the allocations are generated by following this process until all tasks have been assigned to a cluster.

In order to schedule τ_i to processors at a minimum cost while still respecting both the reliability and deadline constraints, resource assignment is called for the current task, for instance τ_{10} of cl_1. This procedure first checks the assignment of the current task's cluster to any processor. If there is a processor, resource assignment, schedules this task to that processor. Then, computes AST_{10}, and updates LFT with its unassigned predecessors. Otherwise, resource assignment tries to find the best admissible assignment for that task. Therefore, the sub-reliability requirement of the cluster should first be computed. In this example, there are three possible processor assignments for this cluster and the assignment of rs_3 with its minimum cost is the best admissible assignment among them. The task then calculates AST_{10} and updates

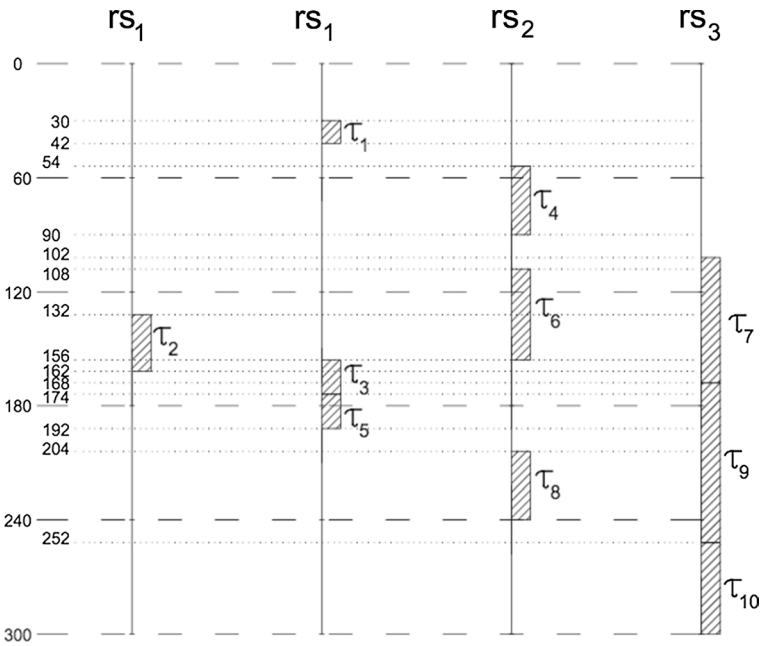

Fig. 2 Sample workflow schedule map

LFT with its unassigned predecessors, i.e., τ_9. The rest of the allocations are generated by following this process until all tasks have been assigned to a processor.

Figure 2 presents the schedule map of the CbCP algorithm for the sample workflow. The boxes indicate the tasks and the numbers outside the boxes denote the task numbers. The execution start and finish time of each task can be specified according to the time bar. The τ_{entry} and τ_{exit} with zero computation and communication time have no effect on the scheduling procedure and are not shown on the schedule map.

The total execution time, reliability, and cost are 300, 0.9402, and \$32 respectively.

5 Performance evaluation

This section presents the results of simulations of the Clustering based on Critical Parent algorithm. Schedule Cost and System Reliability are appointed as assessment criteria. The schedule cost estimates the monetary cost, while system reliability determines the performance of each algorithm.

5.1 Experimental workflows

The performance of a scheduling algorithm should be measured on a sample workflow for evaluation. This can be accomplished, for example, by utilizing a random

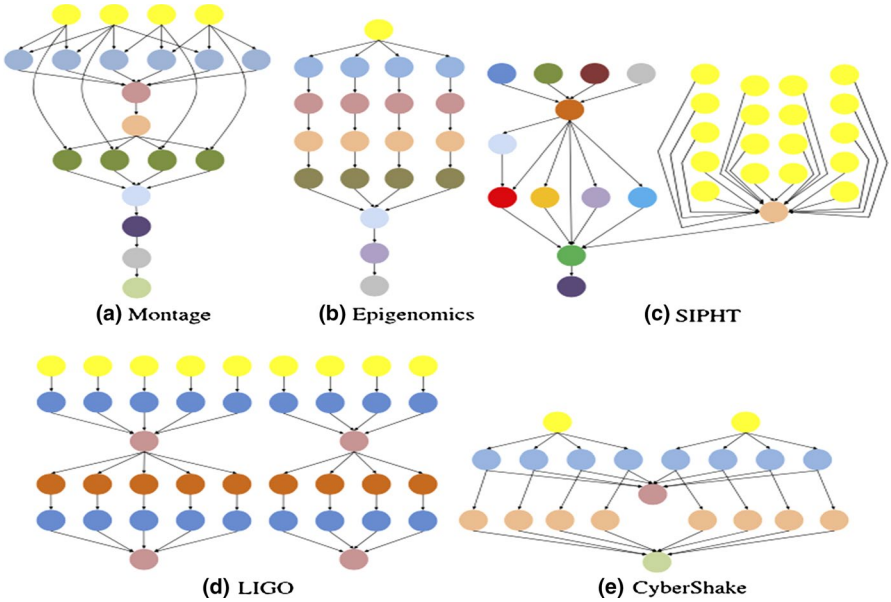

Fig. 3 The structure of five realistic scientific workflows [35]

DAG generator to produce a variety of workflows with different characteristics or by employing a library of realistic workflows which are applied in the scientific or business community.

In one of the initial works in this area, Bharathi et al. studied five realistic workflows, namely Montage, CyberShake, Epigenomics, LIGO, and SIPHT [35]. These graphs are related to real scientific workflows in various scientific fields, such as astronomy, earthquake science, biology, and gravitational physics. Figure 3 illustrates the approximate structure of these workflows with a small number of nodes. It should be noted that these workflows have various structural specifications in regard to their composition and main components, such as pipelines, data aggregation, data distribution, and data redistribution. For each workflow, tasks with an identical color are of the same category and can be processed with a common service. [35] provides a thorough characterization for each workflow and describes their structures, data, and computational requirements. The DAX (directed acyclic graph in XML) format of these workflows is available in Bharathi et al.'s website. The current paper chose three sizes of workflows for its experiments: small (about 25 tasks), medium (about 50 tasks), and large (about 100 tasks).

5.2 Experimental setup

The resource type considered in the present research is based on Amazon AWS EC2. Table 4 presents examples and their related leasing prices for a period of

Table 4 VM types used in the experiments

Type	Memory (GB)	Core speed (ECU)	Cores	Cost ($)
m1.small	1.7	1	1	0.06
m1.medium	3.75	2	1	0.12
m1.large	7.5	2	2	0.24
m1.xlarge	15	2	4	0.48
m3.xlarge	15	3.25	4	0.50
m3.xxlarge	30	3.25	8	1.00

60 min. The current work utilized the Pegasus Workflow Generator [35] to generate the workflows.

The approaches similar to the present study are [4, 16]. In [4], the authors presented the DRR algorithm to minimize the redundancy of a parallel application in order to meet an application's deadline and reliability requirements on heterogeneous distributed systems. In [16] the authors presented QFEC+ algorithm to minimize execution cost in order to meet an application's reliability requirements. The essential limitations of DRR and QFEC+ procedures are: 1- the workflow structure is ignored; 2- the sub-reliability requirements of the tasks are high. The latter will increase the need for unnecessary redundancy to satisfy the sub-reliability requirements while the former is crucial when there are high interdependencies among tasks. Nonetheless, QFEC+ tried to tackle the second issue by considering an upper bound in computing the sub-reliability of tasks, which leads to lower number of replicas compared to DRR method.

To compare the current study's simulation outcomes with those from DRR and QFEC+ algorithms, five different deadline intervals and reliability values are defined from tight to relaxed. Firstly, for each workflow, the HEFT strategy calculates the deadline, because the deadline must be greater than or equal to the makespan of a similar workflow scheduled with the HEFT strategy. Different deadline thresholds were then computed by multiplying its deadline by the constant c, where c ranges from 1 to 5. When $c = 1$, the deadline is very tight; however, higher values of c represent more relaxed deadlines. In addition, different reliability thresholds are defined from 0.95 to 0.9 with 0.01 decrements. It should be considered that the reliability of the processors and communication links used in this phase are known. In other words, these values are obtained from tables of reliability given by the Cloud provider.

5.3 Experimental results

A deadline and reliability should be assigned to each workflow for the CbCP scheduling algorithm evaluation. The execution of each workflow is simulated by HEFT [8] in order to obtain the workflow's corresponding deadline and reliability. Moreover, to overcome the difference in the attribution of workflows, the total cost is

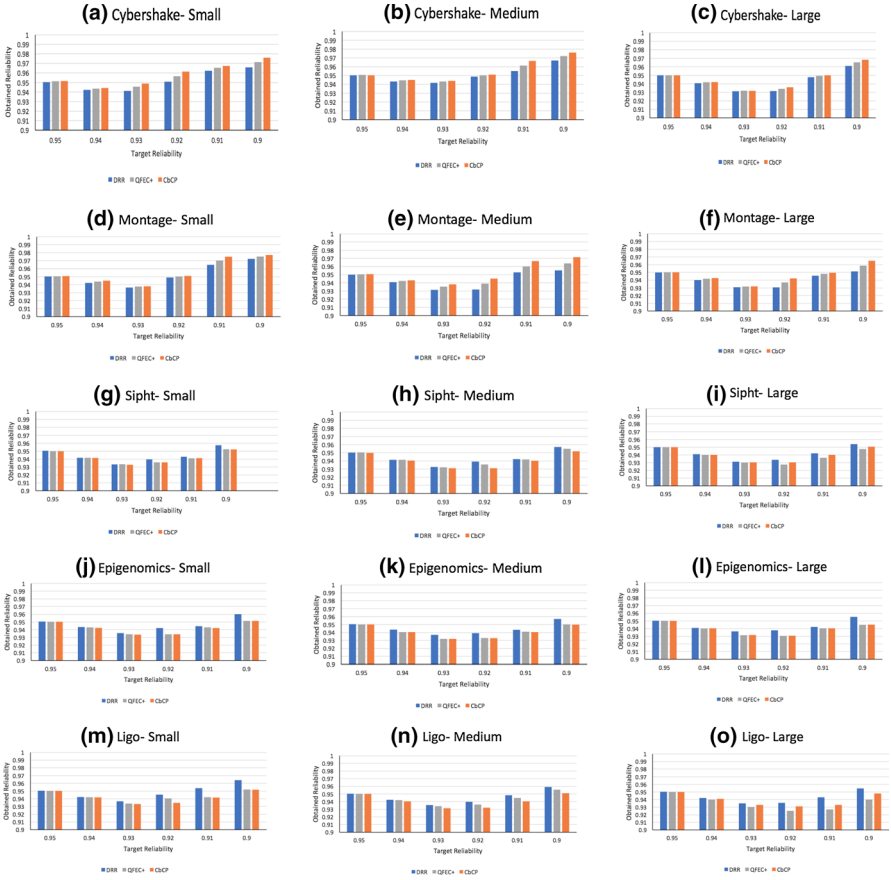

Fig. 4 Obtained Reliability of scheduling workflows with the CbCP, DRR and QFEC+ algorithms

normalized to make the comparison more convenient. Thus, 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 evaluate the CbCP, DRR and QFEC+ algorithms under testing, *Schedule Cost* and *System Reliability* are elected for the evaluation criteria. The schedule cost compares the monetary cost of the three algorithms, while system reliability evaluates the performance of these algorithms.

In the present study's simulation experiments, in all three algorithms, all workflows were successfully scheduled before their deadlines and according to their reliability, with tight deadlines (small deadline factor) and workflow reliability threshold R (equal to 0.95). The system reliability of the three algorithms is compared with respect to their graph characteristics. Figure 4 illustrates the overall experimental outcomes. Compared to DRR and QFEC+ , the current paper's CbCP algorithm achieves noticeable improvement in system reliability in the workflows with high interdependencies among tasks, i.e., CyberShake and Montage workflows. In

these workflows, by increasing the number of processors in DRR and QFEC+ , the average number of links from one processor to another relatively increases and link reliability influences system reliability. On the contrary, while the proposed method schedules each cluster on one processor, the algorithm works very well for workflows with high interdependencies among tasks. In addition, the key limitations of the DRR and QFEC+ algorithms are ignoring the workflow structure, as well as high sub-reliability requirements of all tasks. Therefore, the overrunning reliability values (i.e., $Rel_{schedul} - R$) are usually high for DRR and QFEC+ . In contrast, CbCP's overrunning reliability values are lower in SIPHT, Epigenomics, and LIGO. Nonetheless, QFEC+ tried to tackle the second issue by considering an upper bound in computing the sub-reliability of tasks, which leads to lower number of replicas compared to DRR method. Compared to high-parallelism workflows such as Epigenomics, overrunning reliability values are very low in CbCP and QFEC+. To sum up, CbCP algorithm, has improved the system reliability in CyberShake and Montage. However, in other workflows, e.g., SIPHT, Epigenomics, and LIGO, the system reliability is just satisfied.

Figure 4 demonstrates that an increase in the number of tasks leads to a decline in system reliability, since a rise in the number of tasks correspondingly causes the total execution time of each cluster to increase. Therefore, according to Eq. (14), system reliability lowers.

Figure 5 provides the scheduling costs of all workflows with the CbCP, DRR and QFEC+ algorithms. 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. On the contrary, the DRR procedure employs the HEFT scheduler which usually chooses solutions that do not consider execution costs and iteratively allocates replicas of each task to processors with maximum reliability values until the sub-reliability requirement of the task is met. In addition, the QFEC+ procedure, iteratively chooses available replicas and processors with the minimum execution times for each task until its sub-reliability requirement is satisfied. A quick look at Fig. 5 reveals that, in all workflows, the DRR and the QFEC+ cost results are much higher than those of the CbCP method, because the objective of this scheduling approach is to select the processor just when the workflow makespan and reliability are closest to the specified deadline and reliability. However, the same normalized costs for a relaxed deadline and reliability are obtained in different algorithms, for small, medium, and large size workflows. That is, in the case of flexibility of the deadline and reliability of the workflow, the scheduling process is not affected by structural properties.

As illustrated in Fig. 5, raising the number of tasks from 25 to 100, lowers the normalized cost in Epigenomics while produce same normalized cost in SIPHT and LIGO. Nonetheless, in Montage and CyberShake, an increase in workflow tasks raises the normalized cost. This is due to the structure of these workflows that leads to make up small clusters. Thus, the resource assignment algorithm has to establish many resources, while just a small part of their time slot is utilized.

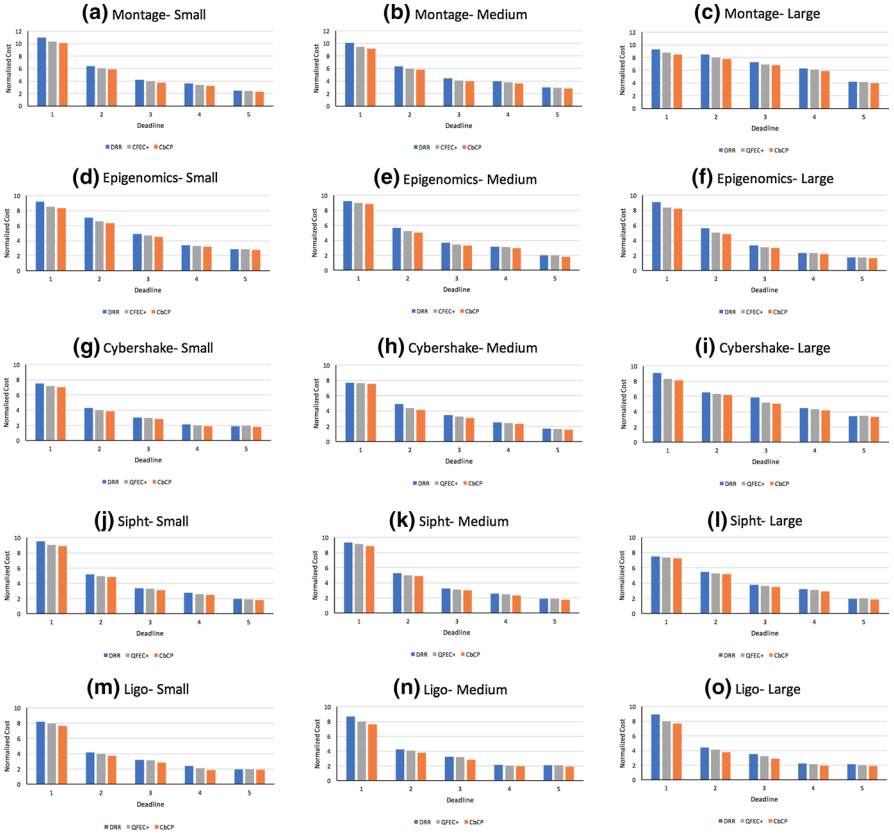

Fig. 5 Normalized Cost of scheduling workflows with the CbCP, DRR and QFEC+ algorithms

6 Conclusions

Cloud computing allows users to attain their desired QoS (e.g. deadline and reliability) by paying a relevant price. For workflow scheduling, the current paper proposes a new algorithm, CbCP, that minimizes the total execution cost while satisfying a user-specified deadline and reliability.

Compared with DRR and QFEC+, the main advantage of CbCP is its capability to obtain lower sub-reliability requirements for clusters. To evaluate the proposed algorithm, the current work utilized synthetic workflows based on real scientific workflows with different structures and different sizes. The results show that CbCP provides the best possible solution when the task graph satisfies a simple condition. Even if the condition is not satisfied, the proposed algorithm provides a satisfactory schedule that is close to the optimum solution with a short computation time.

In heterogeneous service-oriented systems such as Clouds, resources utilize huge amounts of electrical energy. It is important to employ workflow scheduling

to optimize negative impacts of electrical energy, for instance, scaling down computation cost and carbon dioxide, as well as scaling up the reliability of system. In addition to concerning execution cost and reliability, in our future work, we will take into account the energy consumption of resources. Hence, it is essential to propose an efficient technique for reducing energy consumption, as well as satisfying the user QoS constraint requirements in such environment.

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

References

1. Cai Z, Li X, Gupta JND (2016) Heuristics for provisioning services to workflows in XaaS clouds. IEEE Trans Serv Comput 9(2):250–263
2. Zhu X, Wang J, Guo H, Zhu D, Yang LT, Liu L (2016) Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. IEEE Trans Parallel Distrib Syst 99:1
3. Zhou A et al (2016) Cloud service reliability enhancement via virtual machine placement optimization. IEEE Trans Serv Comput 10(6):902–913
4. Zhao L, Ren Y, Sakurai K (2013) Reliable workflow scheduling with less resource redundancy. Parallel Comput 39(10):567–585
5. Qiu W, Zheng Z, Wang X, Yang X, Lyu MR (2014) Reliability-based design optimization for cloud migration. IEEE Trans Serv Comput 7(2):223–236
6. Silic M, Delac G, Srbljic S (2015) Prediction of atomic web services reliability for QoS-aware recommendation. IEEE Trans Serv Comput 8(3):425–438
7. Bajaj R, Agrawal DP (2004) Improving scheduling of tasks in a heterogeneous environment. IEEE Trans Parallel Distrib Syst 15(2):107–118
8. Daoud MI, Kharma N (2008) A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J Parallel Distrib Comput 68(4):399–409
9. Wieczorek M, Hoheisel A, Prodan R (2009) Towards a general model of the multi-criteria workflow scheduling on the grid. Future Gener Comput Syst 25:237–256
10. Yu J, Kirley M, Buyya R (2007) Multi-objective planning for workflow execution on Grids. In: Proceedings of the 8th IEEE/ACM international conference on grid computing, pp 10–17
11. Benoit A, Hakem M, Robert Y (2008) Fault tolerant scheduling of precedence task graphs on heterogeneous platforms. In: IPDPS Miami 2008—proceedings of the 22nd IEEE International symposium on parallel and distributed processing CD-ROM, vol 33, no. December 2007
12. Benoit A, Hakem M, Robert Y (2009) Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems. Parallel Comput 35(2):83–108
13. Girault A, Kalla H (2009) A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. IEEE Trans Dependable Secur Comput 6(4):241–254
14. Zheng Q, Veeravalli B (2009) 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
15. Zheng Q, Veeravalli B, Tham CK (2009) 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
16. Xie G, Zeng G, Li R, Li K (2017) Quantitative fault-tolerance for reliable workflows on heterogeneous IaaS clouds. IEEE Trans Cloud Comput. https://doi.org/10.1109/TCC.2017.2780098
17. Arabnejad H, Barbosa JG (2014) A budget constrained scheduling algorithm for workflow applications. J Grid Comput 12(4):665–679

18. Sakellariou R, Zhao H, Tsiakkouri E, Dikaiakos MD (2007) Scheduling workflows with budget constraints. In: Integrated research in GRID computing CoreGRID integration workshop 2005 Selected Papers, pp 189–202
19. Su S, Li J, Huang Q, Huang X, Shuang K, Wang J (2013) Cost-efficient task scheduling for executing large programs in the cloud. Parallel Comput 39(4–5):177–188
20. Szabo C, Kroeger T (2012) Evolving multi-objective strategies for task allocation of scientific workflows on public clouds. In: 2012 IEEE congress on evolutionary computation, CEC 2012, pp 10–15
21. Kianpisheh S, Charkari NM (2014) A grid workflow quality-of-service estimation based on resource availability prediction. J Supercomput 67(2):496–527
22. Xie G et al (2017) Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems. IEEE Trans Serv Comput. https://doi.org/10.1109/TSC.2017.2665552
23. Qin X, Jiang H, Swanson DR (2002) An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In: parallel processing. 2002. Proceedings international conference on, pp 360–368
24. Benoit A, Hakem M, Robert Y (2009) Optimizing the latency of streaming applications under throughput and reliability constraint. In: Proceedings international conference on parallel processing, pp 325–332
25. Zhao L, Ren Y, Sakurai K (2011) A resource minimizing scheduling algorithm with ensuring the deadline and reliability in heterogeneous systems. In: Proceedings of the international conference on advanced information networking and applications AINA, pp 275–282
26. Deelman E, Gannon D, Shields M, Taylor I (2009) Workflows and e-Science: an overview of workflow system features and capabilities. Future Gener Comput Syst 25(5):528–540
27. Naghibzadeh M (2016) Modeling and scheduling hybrid workflows of tasks and task interaction graphs on the cloud. Future Gener Comput Syst 65:33–45
28. Benoit A, Canon LC, Jeannot E, Robert Y (2012) Reliability of task graph schedules with transient and fail-stop failures: complexity and algorithms. J Sched 15(5):615–627
29. Deldari A, Naghibzadeh M, Abrishami S (2017) CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud. J Supercomput 73:756–781
30. Bittencourt LF, Madeira ERM (2010) Towards the scheduling of multiple workflows on computational grids. J Grid Comput 8:419–441
31. Bittencourt LF, Madeira ERM (2008) A performance-oriented adaptive scheduler for dependent tasks on grids. Concurr Comput Pract Exp 20:1029–1049
32. Topcuoglu H, Hariri S (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13:260–274
33. Girault A, Kalla H, Sighireanu M, Sorel Y (2003) An algorithm for automatically obtaining distributed and fault-tolerant static schedules. In: Proceeding of international conference on dependable systems and networks. pp 159–168
34. Ranaweera S, Agrawal DP (2000) A task duplication based scheduling algorithm for heterogeneous systems. In: Proceedings 14th international parallel and distributed processing symposium 2000. IPDPS 2000. pp 445–450
35. Bharathi S, Chervenak A, Deelman E, Mehta G, Su MH, Vahi K (2008) Characterization of scientific workflows. In: 2008 3rd Work. Work. Support large-scale sci. work. no. June 2014