CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud

Arash Deldari[1] · **Mahmoud Naghibzadeh**[1] ·
Saeid Abrishami[1]

Abstract Workflows are adopted as a powerful modeling technique to represent diverse applications in different scientific fields as a number of loosely coupled tasks. Given the unique features of cloud technology, the issue of cloud workflow scheduling is a critical research topic. Users can utilize services on the cloud in a pay-as-you-go manner and meet their quality of service (QoS) requirements. In the context of the commercial cloud, execution time and especially execution expenses are considered as two of the most important QoS requirements. On the other hand, the remarkable growth of multicore processor technology has led to the use of these processors by Infrastructure as a Service cloud service providers. Therefore, considering the multicore processing resources on the cloud, in addition to time and cost constraints, makes cloud workflow scheduling even more challenging. In this research, a heuristic workflow scheduling algorithm is proposed that attempts to minimize the execution cost considering a user-defined deadline constraint. The proposed algorithm divides the workflow into a number of clusters and then an extendable and flexible scoring approach chooses the best cluster combinations to achieve the algorithm's goals. Experimental results demonstrate a great reduction in resource leasing costs while the workflow deadline is met.

✉ Mahmoud Naghibzadeh
naghibzadeh@um.ac.ir

Arash Deldari
Arash.Deldari@stu-mail.um.ac.ir

Saeid Abrishami
s-abrishami@um.ac.ir

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

Ⓐ Springer

1 Introduction

Nowadays, the demand for processing data-intensive and compute-intensive tasks is ever increasing in different fields of scientific applications. As the latest means of utility computing in distributed systems, the concept of cloud computing has drawn much attention from researchers, not only in academics but also in commercial and industrial fields. The most recent emerging trend in this context and one of the greatest features of distributed systems, such as grid and especially the cloud is the on-demand accessibility of processing resources with variable sizes and capabilities. Utility grids employ the monetary concept in a way that storage and processing resources with different quality of service (QoS) characteristics can be provided at different prices [1]. The QoS requirements of users in these systems are guaranteed through user and service provider contracts called service level agreements (SLAs). Considered as an extension of utility grids, cloud computing uses pay-as-you-go and quality of service concepts to offer the utility grid features over the Internet. Instead of investing in required resources and their maintenance, one can lease resources as required. Moreover, the on-demand and pay-as-you-go usage of cloud resources makes this infrastructure highly scalable and cost effective. Through a virtualization process, storage and processing resources are presented on the cloud as separate physical infrastructures, which is an important cloud feature. Therefore, the virtual environment provided by the cloud is entirely independent of other environments and physical hardware [2]. This feature also allows service providers to offer customized environments based on user requirements. Accordingly, the cloud can be considered as an appropriate platform for executing loosely coupled applications consisting of cooperative tasks, such as scientific workflows which may require high computing power far beyond a single processing resource.

The workflow model is used in numerous scientific applications in distributed systems, including those for chemistry, computer science, physics, and biology, etc. One method to describe a workflow is the directed acyclic graph (DAG), in which each task is dependent on the data produced by its predecessors. Data communications between tasks create precedence constraints on the workflow. In this model, the vertices represent the tasks and the edges represent control and data dependencies. In more general cases, some workflows can be modeled as hybrid DAGs composed of tasks and super-tasks in which tasks within super-tasks can interact during execution [3]. To execute this type of applications, many workflow management systems have been designed by projects such as Pegasus [4], GrADS [5], and ASKALON [6] that describe, manage, and execute workflows on the grid. The latest version of these workflow management systems also supports the execution of workflows on the cloud platform.

Workflow scheduling is considered as the process of mapping tasks on processing resources so as to satisfy some performance metrics [7]. Like all task scheduling issues, optimal workflow scheduling is a well-known NP-complete problem. Hence, most task scheduling algorithms attempt to overcome this issue by proposing heuristic and meta-heuristic methods. Most workflow scheduling techniques on the grid try to minimize

the workflow execution time (i.e., makespan) [8,9]. In the pay-as-you-go cloud model, execution cost is one of the most important factors considered in scheduling methods, in addition to that of the workflow makespan. Therefore, a proper trade-off between two of the most important QoS factors, execution cost and makespan renders the problem of scheduling workflows on the cloud even more challenging.

On the other hand, the widespread development of multicore processor technology has caused service providers to choose these kinds of processors as their infrastructure. Consequently, workflow execution on the cloud must consider scheduling on multicore processors in such a way that the user-defined deadline is met, and the monetary costs of the workflow execution are also minimized. Due to the higher leasing costs of multicore resources, the execution costs and utilization of leased resources gain higher importance. Most of the current studies conducted in this context have not considered the multicore resources available on the cloud and their utilization.

The current study proposes the cluster combining algorithm (CCA), a new static scheduling algorithm that attempts to overcome the problem of workflow schedul-ing on the multicore cloud. The main goal is to minimize the monetary costs of workflow execution while meeting a user-defined deadline. The proposed approach consists of two main phases; first, the workflow is clustered by a primary clus-tering algorithm. The primary clustering phase of the CCA algorithm considers a sequence of related tasks for each cluster. We have assumed that for each cluster after the primary clustering phase a single-core processing resource is leased. This is only an assumption and not a fact. The second phase of the algorithm chooses the best combination among the available cluster combinations via a novel scoring approach and maps cluster tasks on multicore processing resources using a step-by-step approach. An extendable and flexible scoring approach is presented for the combination phase. In this phase of the algorithm, the best available cluster com-bination candidate meeting the deadline and reducing the workflow execution cost achieves a score higher than the other combinations and so is chosen. This scor-ing method also considers the serial and parallel execution of the resulting clusters and selects the best execution method based on the achieved score. This approach employs a new concept called time overlap to decide how to execute the combined clusters. The clusters with a high time overlap are executed in parallel while those with no time overlap or a low time overlap are executed in series. Therefore, the par-allel execution of clusters results in a higher score if the combining clusters have a high time overlap. Obtaining a high time overlap in a parallel execution reduces the free time gaps in the schedule map, which directly increases the utilization of the leased resources. If the workflow deadline is violated, the algorithm will attempt to reduce the makespan even by leasing powerful resources and undertaking more free time gaps, both of which increase monetary costs. The second phase of the algorithm is repeated until no other combination of available clusters improves the workflow score.

The main contributions of the present study can be listed as follows:

– Exploitation of multicore platform facilities.
– Consideration of multiple quality of services in the scheduling algorithm.
– Usage of a heuristic method for discerning the best cluster combination.

– Introduction of an extendable and flexible scoring method which allows multiple criteria scheduling.

The rest of the paper is organized as follows. A background on workflow scheduling, resource management, and related work on distributed systems is provided in Sect. 2. Section 3 presents in detail the required system model and basic definitions. The proposed algorithm is described in Sect. 4. Section 5 presents the simulation results, and the execution time's performance evaluation and the cost of the proposed algorithm. A brief discussion on algorithm's performance is provided in Sect. 6. Concluding remarks and intended future works are covered in the last section.

2 Related work

The cloud offers four types of services, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), and Network as a Service (NaaS). Some of these services are broader and can be considered as platforms for other services. For example, Financial Software as a Service (FSaaS) and Risk Visualization as a Service (RVaaS) are grouped with SaaS, as in [10,11]. Users can benefit from these services; quality of service (QoS) parameters can be met based on a Service Level Agreement (SLA) between the user and the service provider on a pay-as-you-go plan.

Arabnejad and Barbosa [12] have classified workflow scheduling algorithms in this area into two main categories: QoS optimization and QoS constrained. The QoS optimization algorithm attempts to optimize all QoS parameters in the schedule map. In this category, some research has been conducted to provide a balance between QoS parameters such as time and cost in [13–16]. The QoS constrained algorithms tried to optimize some QoS parameters while meeting other user-defined QoS constraints. For instance, this category can optimize the cost of a workflow execution on a distributed system while taking care not to violate a user-defined deadline, as in [1]. On the other hand, workflow scheduling algorithms can be classified into static and dynamic approaches. In the static mode, the schedule map is created before execution and does not change during execution. In contrast, dynamic scheduling algorithms do not have any previously made plan before runtime. Instead, during runtime, the scheduler plans according to available resources.

Dynamic scheduling methods can be classified into two categories: online and batch. In the online mode, as soon as a task is ready, it is scheduled on a resource which leads to the earliest finish time, as in [17]. On the other hand, the batch mode gathers the ready tasks in a set and schedules them on resources in a best-effort manner. The two most well-known algorithms in this context are Min–Min [18] and Max–Min [18]. The Min–Min method selects the task with the minimum earliest finish time and schedules it on a resource that provides the earliest completion time. Similarly, the Max–Min algorithm selects the task with the maximum earliest finish time.

Some comparative studies consider that static scheduling methods outperform dynamic ones from different points of view in most cases [19]. The main reason for this superiority is using available prior information, both at the task and workflow level.

The present research is a QoS constrained heuristic on the static scheduling of scientific workflows on the IaaS cloud. Time and cost can be considered as two of the most important user quality of services. Therefore, a proper time/cost trade-off is very critical in scheduling algorithms [12].

Generally, optimal task scheduling pertains to the NP-complete class. To this aim, heuristic and meta-heuristic methods attempt to address the issue in polynomial complexity [20–23]. Another category of studies has investigated these NP-complete problems using mathematical formulations [24, 25].

Scheduling on distributed systems such as grid and cloud, has been investigated in a wide range of studies, such as [8,9,26–28]. Numerous early studies on distributed systems proposed reducing execution time (i.e., makespan). For instance, the HEFT algorithm introduced by Topcuoglu et al. [20] is one of the most famous algorithms in the static list scheduling context on heterogeneous processors. This method selects the task with the highest upward rank and assigns it to a processing resource, which leads to the minimum earliest finish time of the selected task. The CPOP algorithm, which uses the summation of the upward and downward rank as the priority of each task, is also introduced in the current research. A wide range of heuristics is dedicated to the extension of this well-known method as discussed in [12,29,30].

The Dominant Sequence Clustering (DSC) algorithm proposed by Yang and Gerasoulis [31] is one of the well-known clustering scheduling algorithms based on prioritizing directed acyclic graphs. This method selects a task with the highest priority in each step and combines it with one of its parents' clusters. The Path Clustering Heuristic (PCH) proposed in [22] is also a cluster-based static scheduling algorithm on the grid. The main objectives are maximizing the throughput and utilization of resources and reducing runtime. A similar method was also proposed for scheduling multiple workflows in [32], which maximizes the fairness between processes while minimizing the required scheduling time. A two-stage procedure for scheduling on multiprocessors is proposed by Sarkar [21]: (1) a primary clustering phase based on edge zeroing with the assumption an infinite number of processors exist and (2) the aggregating and scheduling of these primary clusters to meet the number of available processors.

HPC application scheduling on the cloud data centers with energy efficiency modeling is considered in [33]. An increase in energy consumption decreases the profits of service providers and drastically affects environmental pollution. A resource provisioning heuristic for the cloud data centers is proposed in [34] by setting fixed utilization thresholds. The main objective of this research is reducing energy by the dynamic adaptation of virtual machine allocation during runtime and also by turning idle nodes to the sleep mode.

Much research [35–38] has investigated the issues of considering load balancing, performance, accessibility, and reliability in resource management and scheduling on the cloud. The pricing model used on the cloud is one of the main differences between the cloud and the grid. Hence, execution expenses, in addition to execution time, are considered as an important QoS factor in most scheduling algorithms on the cloud.

The SPSS algorithm proposed by Malawski et al. [39] is a static scheduling algorithm for multiple workflows on the IaaS cloud that tries to minimize costs with respect to the specified deadline. Firstly, it allocates a sub-deadline to each workflow task and

chooses tasks according to their sub-deadline. These are then mapped on the cheapest possible resource that minimizes the start time of the task. However, this research has not considered the available multicore processing resources of the cloud.

The IC-PCP algorithm was proposed by Abrishami et al. [7] to schedule scientific workflows on the IaaS cloud, an extension to their previous PCP method [1] for scheduling workflows on the grid. This approach minimizes execution costs by considering time constraints using the Partial Critical Path concept. Therefore, a sequence of tasks is mapped onto a processor for execution. Furthermore, they proposed the Budget-PCP [40] which minimizes execution time with respect to cost constraints. This approach distinguishes the sequence of tasks well, but does not recognize the relationship between the sequences. Accordingly, this method cannot function properly for scheduling on multicore processors.

Poola et al. [41] presented a critical path-based workflow scheduling algorithm on the cloud which considers robustness and fault-tolerance with time and cost constraints by using three different resource allocation policies. This method solves the problem of uncertainties, such as performance variations and failures in cloud environments, by adding slack time based on deadline and budget constraints. However, this method does not take into account the available multicore processing resources in the cloud.

Introduced by Bittencourt and Madeira [2], the Hybrid Cloud Optimized Cost (HCOC) algorithm schedules multiple workflows on private and public clouds and also considers multicore processing resources. The general purpose of this method is to schedule workflow on the private cloud. Hence, if the deadline is not met, resources are leased from a public cloud to meet the specified deadline while trying to minimize the renting cost. This method's main deficiency occurs when the workflow does not meet the deadline. As a result, the algorithm functions very slowly, mainly because, in each iteration, more tasks are selected for testing on a higher number of processing resources.

3 System model and basic definitions

This section presents the fundamentals of the current work's scheduling algorithm, such as application, resource, and pricing models.

3.1 Application model

The workflow model is one of the most successful paradigms for programming scientific applications on distributed infrastructures, such as the grid and the cloud. Bags of tasks is another successful application that can be covered by this model [42,43]. The present study's scheduling algorithm focuses on scheduling scientific workflows on the cloud.

Since a workflow's tasks are optionally interconnected, the application model used for a scientific workflow is a Directed Acyclic Graph $G = (V, E)$, in which $V = \{v_i | i = 1, \ldots, V\}$ denotes the set of tasks of the workflow, and $E = \{e_{i,j} | (i, j) \int \{1, \ldots, V\} \times \{1, \ldots, V\}\}$ shows the edges between the vertices. Also, each edge has a weight which denotes the precedence constraint and the amount of data commu-
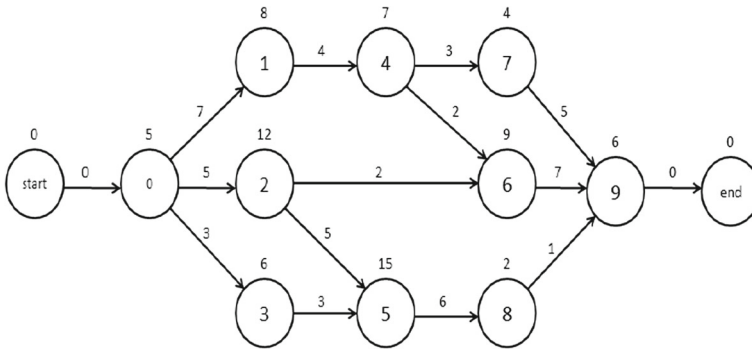
Fig. 1 Sample workflow

nication between tasks v_i and v_j. The amount of data communication between tasks is predetermined and known in advance. Therefore, for the computation of a task to begin, all the data needed for its predecessors must be received and ready. Since the algorithm requires a graph with individual entry and exit nodes, a v_{entry} and v_{exit} with zero processing time and zero communication have been added to the DAG. Many techniques, such as analytical benchmarking, code profiling, statistical prediction, and code analysis, have been used to estimate the execution time of a task on an arbitrary resource [1]. Therefore, the average computational capacity of each task is assumed to be known a priori.

With each workflow DAG, a deadline and/or maximum available budget are submitted by the user. Hence, a sub-deadline is computed for each task which must be observed so that the workflow meets the deadline. The focus of the present research is scheduling a single workflow on the cloud and observing a user-defined deadline. However, simple techniques can be used to combine multiple workflows into a single DAG that share a common deadline and/or an overall available budget.

Figure 1 shows a small sample workflow consisting of 10 nodes and two dummy nodes, *start* and *end*. The numbers inside the nodes denote the task numbers and the numbers above the nodes and the edges denote the estimated average execution time and the estimated average communication time, respectively, in minutes. The two dummy nodes are added so that the workflow has a single entry, and also a single exit node and the computation and communication times of the two dummy nodes are considered as zero. For instance, for the processing of task 5 to begin, all the data needed from tasks 2 and 3 must be received. Similarly, all the data from tasks 6, 7, and 8 must be ready to begin the computation of task 9. The makespan of the workflow, in this case, will be the finish time of the *end* node which is equal to the finish time of task 9.

3.2 Resource model

The IaaS cloud model used by the current work offers a set of virtualized processing resources to its users. To model the cloud service provider S, the triple $IaaS_S = (SP, RL, BP)$ is employed, which denotes the service provider's name, a list of

available multicore resources and resource attributes, and billing time periods. The processing resources on the cloud are modeled as inconsistent heterogeneous resources in such a way that different tasks may achieve different performances on the same processing resource. In addition, each multicore processing resource consists of a set of homogeneous cores. The present study's scheduling algorithm uses a resource model which consists of R heterogeneous resources, in which all resources can belong to a single service provider or different service providers. Each multicore resource node i consists of a set of homogeneous cores. To model cloud resource R, $\mathrm{RL}_R = (C, \mathrm{PW}, \mathrm{PR}, M)$ is employed which, respectively, denotes the number of cores of each resource, the computational capacity of each core, the leasing price for each resource billing unit, and also the available memory for each instance. Although there are many QoS attributes in this context, only execution time and cost, respectively, is *QoS (T, C)*, are considered the most important. *Bandwidth* is also a square matrix that defines the communication bandwidth between resources; this is given by the service provider and is equal among the resources offered by the specific service provider.

3.3 Pricing model

One of the available pricing models used by most commercial IaaS cloud service providers is based on a pay-as-you-go manner. In this model, users are charged by the amount of service demanded and without any up-front investment. In other words, the user is charged based on overall leasing time intervals used for the execution of the workflow, even if the user does not completely use the leased time intervals. Because the present work's scheduling algorithm divides the workflow into clusters and each cluster is executed on a separate resource, the execution cost of each cluster can be separately calculated. To compute the cloud's data transfer cost of the workflow, namely $\mathrm{TC}(e_{i,j}, a, b)$ between resource a that is processing task i and resource b that is processing task j, it is assumed that the amount of communication inside the cloud is free, and each service provider (e.g., Amazon) clearly specifies the data communication fees from/to cloud service providers outside the current cloud. Therefore, this parameter is modeled as $P_s^{\mathrm{out/in}}$, which denotes the data transfer price to and from instance s in \$ per GB for non-local data communications.

3.4 Basic definitions

The current study assumes that the average execution time of each task on a single-core processor and the communication capacity between all the workflow tasks is a priori. Hence, the transfer time of each communication can be computed by dividing the data communication volume by the available bandwidth among the service providers' processing resources executing data dependent tasks.

Also, the Minimum Execution Time $(\mathrm{MET}(v_i))$ and the Minimum Transfer Time $(\mathrm{MTT}(e_{i,j}))$ are defined as follows:

$$\mathrm{MET}(v_i) = \min_{s \in S_i} \mathrm{ET}(v_i, s) \tag{1}$$

$$\text{MTT}(e_{i,j}) = \min_{r \in S_i,\, s \in S_j} \text{TT}(e_{i,j}, r, s) \tag{2}$$

Here v_i is the ith task, S_i and S_j are the set of available services, and $e_{i,j}$ denotes the communication edge between tasks v_i and v_j. Also, ET and TT are acronyms for execution time and transfer time, respectively. Using the following definitions, the Earliest Start Time (EST) is computed as follows:

$$\text{EST}(v_{\text{entry}}) = 0 \tag{3}$$

$$\text{EST}(v_i) = \max_{v_p \in v_i'\, parents} (\text{EST}(v_p) + \text{MET}(v_p) + \text{MTT}(e_{p,i})) \tag{4}$$

Also the *latest finish time* of each task is defined as the latest time at which the task's computations must be completed so that the workflow meets the specified deadline as shown in (5) and (6):

$$\text{LFT}(v_{\text{exit}}) = deadline \tag{5}$$

$$\text{LFT}(v_i) = \min_{v_c \in v_i'\, schildren} (\text{LFT}(v_c) - \text{MET}(v_c) - \text{MTT}(e_{i,c})) \tag{6}$$

Therefore, the overall execution time or makespan of the workflow is defined as the time between v_{entry} and the completion of v_{exit}.

4 The proposed workflow scheduling algorithm

The main objective of the present research is the QoS constrained static scheduling of scientific workflows on the multicore IaaS cloud platform.

The widespread availability of multicore processing resources on the cloud poses a significant challenge for the scheduling of workflows. The issue of a multicore platform adds some difficulties to this scheduling problem. One of the main purposes of using these processors is parallel computing. The data communications and precedence constraints between workflow tasks make optimal scheduling of the workflow more difficult to achieve. Therefore, the workflow can be divided into clusters that can be executed in parallel or serial. The concept of clustering the workflow has been studied in different scheduling algorithms [22,32]. The current research proposes an algorithm which combines existing clusters according to certain criteria.

The higher leasing cost of multicore processors compared to that of single cores significantly increases the importance of cost in the multicore cloud and the utilization of available cores. In addition, execution cost and makespan in the cloud environment create conflicting side effects; that is reducing the makespan requires higher processing power resources, thus resulting in higher leasing fees. On the other hand, resources with desirable lower computing capacity and lower leasing prices increase the makespan. Therefore, a proper trade-off between these two concepts is vital.

In order to choose the optimum cluster combination in each combination step, a scoring function has been introduced. The proposed scheduling algorithm is designed
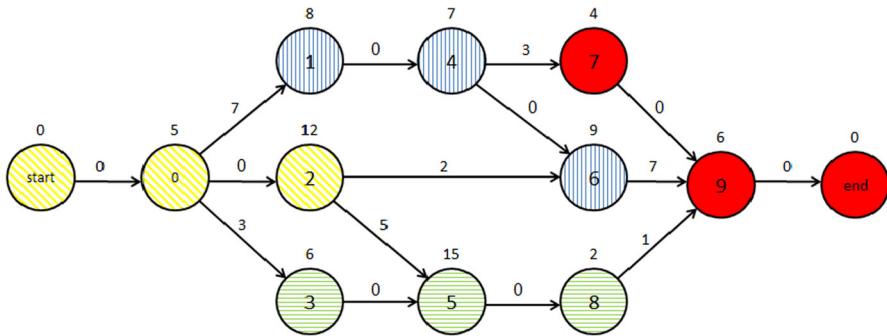
Fig. 2 Sample workflow after the primary scheduling phase

in a way that different criteria in the workflow scheduling context, such as energy efficiency and load balancing among others, can be investigated by the criteria in the scoring function. Thus, the criteria may be applied in the cluster combination phase, and the scheduling will evolve in a way that those criteria will be enhanced during each combination phase. The scoring method uses the new concept of time overlap to decide which clusters should be combined and which ones should remain unchanged. Time overlap denotes the amount of time intersection between the clusters. This concept is also used in deciding the execution manner of the combined clusters. Executing two clusters with a high time overlap in parallel reduces the free time gaps in the available leased time periods. Reducing the free time gaps in each step lessens the overall needed time intervals which directly affect the execution cost of the workflow. To compute the time overlap for each pair of clusters, a time period for the execution of each cluster must be defined. Having computed the EST and LFT of each task of the workflow the Earliest Finish Time (EFT) of each task is also computed as follows:

$$EFT(v_i) = EST(v_i) + MET(v_i) \tag{7}$$

Therefore, as shown in (8) and (9), the EST and EFT can also be defined for each cluster according to the task in each cluster:

$$EFT(C_i) = Max(T_j.eft) \quad \text{s.t} \quad T_j \in C_i \tag{8}$$

$$EST(C_i) = Min(T_j.est) \quad \text{s.t} \quad T_j \in C_i, \tag{9}$$

where C_i denotes the ith cluster and T_j denotes the jth workflow task. Hence, by considering EST of each cluster as the start time of the cluster and the EFT as the finish time of the cluster, the minimum execution time interval for each cluster can be computed. The amount of time overlap between each pair of clusters can be calculated based on the minimum execution time interval of the clusters. Considering the workflow in Fig. 2, the EST of cluster1 composed of nodes start, 0, and 2 is 0 and the EFT is 17. Accordingly, the EST and EFT of cluster2 composed of nodes 1, 4, and 6 are 12 and 36. Therefore, the time overlap interval of these two clusters is between 12 and 17.

Executing two clusters in serial is only possible if the LFT of all the combining cluster tasks is met. Even if only one task misses its LFT, then the workflow deadline will be missed. The general intention in the serial execution of clusters is making the maximum use of the rented time periods of the processing resources.

The CCA algorithm consists of two main phases:

– In the first phase which is called *pre-clustering*, the workflow is divided into primary clusters and a single-core processing resource for the execution of each cluster is considered.
– The second phase is called *combination and mapping,* performs the main part of the algorithm. In this phase, a priority is assigned to each of these primary clusters and the clusters are combined in such a way that the total execution cost of the workflow is reduced, the makespan meets the defined deadline, and the utilization of the processing resources is maximized. Hence, after combining these clusters, the scheduling algorithm must decide the appropriate processing resource for the execution of the resulting cluster. After each cluster combination is fulfilled, a mapping phase is conducted which maps the tasks of the resulting cluster on the cores of the resource. Furthermore, the combination of two clusters zeros their intra-cluster communications which affects the attributes of the tasks. Therefore, attributes, such as the EST and LFT, must be recalculated after each combination.

Depending on the structure of the workflow, different clustering algorithms can be applied in the primary phase. Each task can even be considered as a cluster in the primary clustering phase. In this way, the proposed approach starts by combining these individual tasks, acting in a way as a clustering algorithm with special consideration of multicore processing resources. The default algorithm that is used in the primary clustering phase is the algorithm proposed by Bittencourt and Madeira [32]. The main advantage of this algorithm is that each primary cluster consists of nodes whose predecessors have already been scheduled or are to be scheduled along with them. In the first phase of the algorithm, it is assumed that a single-core processing resource is considered for executing each cluster. Therefore, k single-core processing resources are considered for executing k primary clusters after executing the first phase of the algorithm.

The second phase of the CCA, which is considered as the main phase of the algorithm, consists of combining the primary clusters generated in the first phase. In this research, we have assumed that reducing workflow's makespan has no advantages for the users and completing each workflow no later than its deadline suffices. Therefore, this method focuses on reducing the costs regarding not to pass the user-defined deadline. However, this assumption is not necessary in all cases. The critical issue in this phase is to choose the best combination from among the available combinations. To this aim, a scoring function is proposed that assigns a score to each available combination, with the best combination achieving the highest score. This scoring function operates in such a way that, by choosing the best available combination, the costs of executing the workflow is minimized, and the user-defined deadline is met. This function evaluates a serial and also a parallel execution for each pair of clusters and chooses the best execution mode based on the workflow score achieved. Therefore, this method not only chooses the best available combination, but also determines the execution mode of the resulting cluster and the required processing resource in

terms of the number of cores. This combination is repeated until no other combination improves the overall score of the workflow. In addition, to regulate the cluster combination phase, a priority to each cluster is assigned. In order to assign a priority to each cluster in this phase of the algorithm, each task must be assigned a priority. To assign a priority to each workflow task the *upward rank* computed in the HEFT algorithm [20] must be employed. This rank determines the critical path distance of each task to the exit task and is defined as follows:

$$rank(n_i) = \text{ET}_i + \max_{n_j \in succ(n_i)} (\text{TT}_{i,j} + rank_u(n_j)) \tag{10}$$

$$rank_u(n_{\text{exit}}) = w_{\text{exit}} \tag{11}$$

where ET_i denotes the execution time of the ith task and $\text{TT}_{i,j}$ represents the communication time between tasks i and j. Furthermore, $succ(n_i)$ shows the immediate successors of the ith task.

The *upward rank* of the HEFT algorithm has a number of advantages, which make it a worthy candidate for the prioritizing phase. The simplicity of the *upward rank* can be considered as its first advantage. Secondly, this distance takes into account the critical path of the workflow and the communication time between each pair of nodes in addition to their computation time. Moreover, this method guarantees that every node of the workflow has a higher priority over its successors.

The priority of each cluster is considered as the maximum priority of the cluster's tasks, as follows:

$$priority(C_i) = \text{Max}(T_j.priority) \quad \text{s.t} \quad T_j \in C_i, \tag{12}$$

where C_i denotes the ith cluster and T_j denotes the jth workflow task. By assigning a priority to each cluster of the workflow, this method combines clusters with higher priorities earlier than other clusters, which gives regularity to the combination phase. If no priority is given to each cluster, then the cluster combination in the early workflow levels (i.e., clusters with higher priority tasks at the beginning of the workflow) might take place after the cluster combination in the later levels. Hence, clusters combined in the early stages of the algorithm may be affected. Therefore, this priority orders the combination of clusters in such a way that those with higher priorities are combined first. In other words, after a cluster combination is performed in the higher levels (i.e., clusters with lower priorities), there is no combination done in the lower levels.

To calculate the combination score of each cluster pair, the execution cost and makespan of the workflow after the desired combination must be computed. Hence, it is necessary that the schedule map of the resulting cluster's tasks on the processing resource be determined so as to compute the workflow execution cost and time. A schedule map must be created for the parallel execution of the combining clusters, and one schedule map must be constructed for the serial mode. To this aim, the HEFT algorithm is extended so that it adapts to the CCA. Accordingly, the precise procedure of task mapping on the processing resource is as follows: among the two clusters' tasks, the task with the highest priority is chosen and mapped on a core on the processing resource, which leads to the earliest finish time of the desired task. This mapping

procedure is repeated for all tasks of the combining clusters until all are scheduled. Such a mapping procedure reduces the free time gaps in each core and, more broadly, in the schedule map of the workflow.

A pruning technique is conducted to reduce the computation capacity of the CCA algorithm which eliminates the score computation and scheduling procedure of the infeasible cluster pairs. Therefore, the parallel combination score and parallel execution of cluster pairs without time overlap drastically increases free time gaps in the schedule map and so will not be computed. Besides, cluster pairs with a distance in execution time intervals of more than one leasing period will not be considered for serial execution nor will their score be calculated.

4.1 Cluster combining scheduling algorithm

The pseudo-code of the CCA workflow scheduling algorithm is shown in Algorithm 1. This algorithm is given a workflow and its related deadline as input. Based on the deadline, the *EST* and *LFT* of all the workflow tasks are computed. In Line 3, the primary clustering phase is executed. After the primary clustering phase is executed, the internal communications of each cluster are zeroed in Line 4. The attributes of the tasks and clusters are updated in Line 5, and the primary scheduling phase is performed in Lines 6 and 7. In line 8, a *priority* is assigned to each cluster which is equal to the maximum *priority* task in the cluster. In other words, the combination of clusters is only performed within the *Cluster_Subset*, which consists of all feasible candidate clusters for a combination with a *priority* less than the current cluster's *priority* in Line 11. Hence, infeasible cluster candidates are eliminated from the *Cluster_Subset* by a pruning technique to reduce the algorithm's unnecessary computations. The *Calculate_Combination* function chooses the best possible combination candidate for each cluster within the *Cluster_Subset* and checks whether or not this combination improves the score of the workflow. After the fulfillment of each iteration of the algorithm, *Currentbest_Schedule* presents the best possible schedule map in terms of execution cost and with respect to the deadline.

Algorithm 1: Cluster Combining Algorithm
0   Cluster_Combining (*Workflow* , *Deadline*)
1   **Begin**
2   Compute EST and LFT of all the workflows tasks;
3   Cluster the graph; *//Primary clustering algorithm*
4   Update communications; *//zeroing internal communications for each cluster*
5   Compute *EST* and *LFT* for each cluster;
6   Assign a single core processor to each cluster;
7   Compute the *Currentbest_Schedule* according to current schedule map;
8   Assign a *priority$_i$* to each cluster;
9   **For** i =2 to length(*clusters*) **do**
10   **begin**
11   *Cluster_Subset* = feasible candidate clusters with p*riority* less than
12   or equal to *priority$_i$*;
13   *(Currentbest_Score,Currentbest_Schedule)=*
14   *Calculate_Combination (Current_Schedule, Cluster_Subset)*;
15   **end;**
16   **return** *Currentbest_Schedule;*
17   **End.**

```
Algorithm 2: Calculate_Combination function
0    Calculate_Combination(Current_Schedule,Cluster_Subset)
1    Begin
2      Best_Schedule = null;
3      Best_Score = 0;
4      Combination_Done= false;
5      For each pair of (C₁ , C₂) s.t. C₁ , C₂ ϵ Cluster_Subset do
6          begin
7            New_Schedule =  Serialcombination(C₁,C₂, Current_Schedule);
8            Serial_Score = Calculate_Score (New_Schedule ,Old_Schedule );
9            New_Schedule =  Parallelcombination(C₁,C₂, Current_Schedule);
10           Parallel_Score = Calculate_Score (New_Schedule ,Old_Schedule );
11           Score = Max (Parallel_Score, Serial_Score);
12           If ( Score > Best_Score ) then
13               begin
14                 Best_Score = Score;
15                 Best_Schedule = new_schedule;
16                 Combination_Done = true;
17               end;
18         end; //the best combination in the Cluster_Subset is considered
19     If (Combination_Done == true) then
20             begin
21               Currentbest_Score = Best_Score;
22               Currentbest_Schedule = Best_Schedule;
23               update Cluster_Subset;
24               update the priorities of the resulting cluster and it's successors;
25               Calculate_Combination(Cluster_Subset);
26             end;
27   return Best_Score, Best_Schedule;
28   End;
```

```
Algorithm 3: Calculate_Score function
0    Calculate_Score ( New_Graph , Old_Graph )
1    Begin
2    Old_Makespan = Old_Graph.makespan();
3    New_Makespan = New_Graph.makespan();
4    Old_Cost = Old_graph.cost();
5    New_Cost = New_Graph.cost();
6    If (New_Makespan <= Deadline && Old_Makespan <= Deadline) then
7        Score = Old_Cost – New_Cost;
8    Else if ( New_Makespan > Deadline  && Old_Makespan <= Deadline ) then
9            Score = -infinite;
10      Else  if ( New_Makespan > Deadline && Old_Makespan > Deadline ) then
11              Score = Old_Makespan – New_Makespan;
12          Else
13              Score = New_Makespan;
14   return Score;
15   End;
```

4.2 The Calculate_Combination function

Algorithm 2 shows the pseudo-code of the Calculate_Combination function. This function is given the *Cluster_Subset* and *Current_Schedule* as input parameters, and it tests all the feasible combinations within the *Cluster_Subset* using a scoring method be discussed in detail in the next section. The Serialcombination function considers the serial execution of the cluster pair and the Parallelcombination addresses their parallel execution. When there is a pair of combining clusters for a parallel operation, processors with more cores are employed. For example, if two clusters use dual core processors for execution, after the combination phase, a quad core processor will execute the resulting clusters. Therefore, instead of two dual core processors being

leased, a quad core processor is rented. According to the scores achieved, the best execution method for each pair of clusters is the one with the highest score. The optimum combination will be the one that improves upon the workflow execution's best score when compared to the *Current_Schedule*. If any combination occurs within the subset, cluster attributes,_such as clusters and_their related priorities should be updated. Moreover, the Boolean variable, *Combination_Done*, is set as *true*, and the Calculate_Combination is called recursively to check for any feasible combinations to improve the score with the new *Cluster_Subset*. It must be mentioned that, with the accomplishment of each combination, the attributes of all the tasks within the combined clusters are affected by the zeroing of the internal communications. Therefore, after each combination phase, all the attributes of the combined clusters and their successor clusters must be updated.

4.3 The Calculate_Score function

The pseudo-code of the *Calculate_Score* function is illustrated in Algorithm 3. First, the makespan and execution cost of both the workflow graph before performing the combination with the current schedule map and the graph after performing the desired combination is computed. The first condition in Line 6 checks whether or not the makespan of both the new graph and the old graph meet the specified deadline. In this case, the combination score will be the difference between the *Old_Cost* and the *New_Cost*. This method combines clusters that reduce the leasing cost when the deadline has been met. In other words, reducing the *New_Cost* increases the combination score. Therefore, the combination with the greatest reduction in the execution cost will achieve the maximum score.

The second condition in this algorithm checks if the combination cannot meet the workflow deadline; if so, it will be identified as ineffective by the assignment of –*infinite* to the score.

If the makespan of both the old graph and the new graph does not meet the workflow deadline, then the difference between the *Old_Makespan* and the *New_Makespan* becomes the combination score. This scoring method ensures the makespan is reduced so as to meet the specified deadline in case of any possibility it won't be fulfilled. Therefore, in this case, the algorithm even shoulders a higher execution cost by leasing resources with more cores, which creates a reduction in the workflow makespan. This continues until the makespan meets the defined deadline. If the deadline is met, the scoring method then focuses on reducing the cost. In other words, whenever the deadline is met, the main goal of the scoring function is to reduce the execution cost.

Lemma 1 *If the scheduling of primary clusters in the first phase of the algorithm without any combination is feasible, then after executing the second phase of the Cluster Combining Algorithm, the scheduling of all the workflow clusters remains feasible.*

Proof Assume that the scheduling of the workflow is feasible regarding the given deadline. It then follows that the scheduling of each cluster is also feasible. On the other hand, the CCA algorithm combines clusters which zeros their intra-cluster data

communications, decreases the EST of some tasks, and leaves the EST of other tasks unchanged. In some cases, the algorithm tries to lower the workflow's execution cost and increases the makespan of the tasks, which is only possible if the LFT of each task is met. Hence, the CCA algorithm will not violate the LFT of the workflow tasks, and so the scheduling of the clusters remains feasible.

Lemma 2 *In the early stages of the algorithm, if the scheduling of the workflow is not feasible according to the given deadline, the CCA method might make workflow scheduling feasible in the later stages by combining the clusters. In other words, as the CCA algorithm advances, infeasible schedules according to the given deadline might be made feasible.*

Proof In the case where workflow scheduling is not feasible regarding the deadline, this algorithm shall lease processing resources with a higher number of cores and even take on free time gaps in the schedule map. Therefore, by executing clusters in parallel and zeroing their intra-cluster data communications, the workflow makespan is reduced, and the deadline perhaps met. However, after meeting the desired deadline, the algorithm will work to minimize the execution cost.

Lemma 3 *If the schedule map of the primary clusters in the first phase of the algorithm meets the user-defined deadline, then the final schedule map of the CCA algorithm will meet the deadline, and the execution cost will be less than or equal to the execution cost of the primary schedule map.*

Proof The CCA algorithm creates a subset for each cluster consisted of feasible combination candidates with time overlap, and the best combination is chosen within the subset using the *Calculate_Combination* function. The best possible combination will be the one with the highest score from the *Calculate_Score* function. The scoring function only considers combinations that do not violate the defined deadline. Furthermore, the combinations that reduce the execution cost the most, will achieve higher scores. The CCA algorithm will reduce the execution cost of the workflow step-by-step by combining the clusters while taking care not to violate the deadline otherwise clusters will not be combined.

4.4 A sample of workflow scheduling

By the usage of an example, the proposed workflow scheduling method is presented in more detail. The scheduling sample considers the workflow shown in Fig. 1. It is also assumed that the defined deadline for this workflow is 47 min.

It has also been supposed that the available processing resources are single cores, dual cores, and quad cores and that their leasing prices for one billing unit are $0.1, $0.2 and $0.4, respectively, as obtained from the Amazon EC2 pricing policy. The length of each billing unit is considered 10 min.

According to the defined deadline, the EST and LFT of all workflow nodes are first computed. The initial phase of the algorithm is primary clustering and updating the graph communications. Figure 2 shows the workflow after primary clustering,
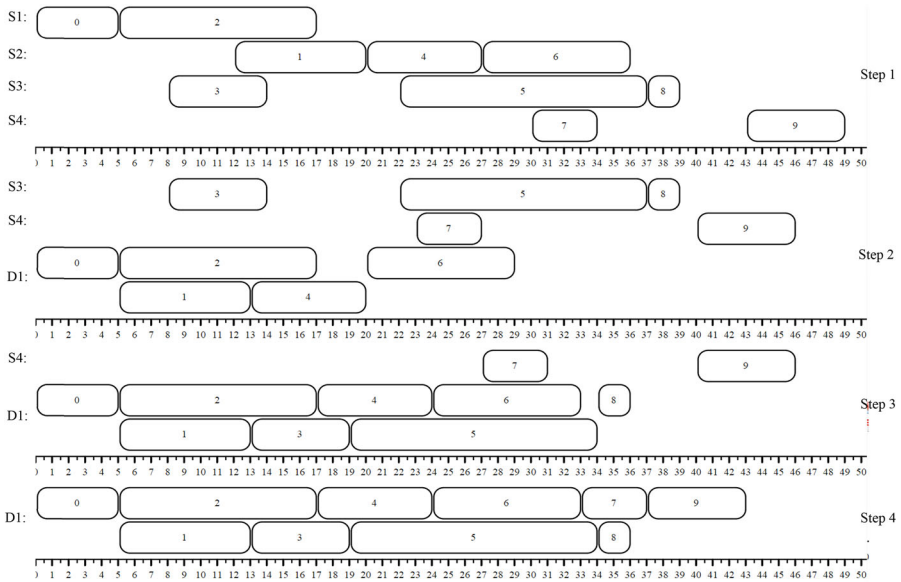
Fig. 3 Sample workflow schedule map

which consists of four clusters; each with three nodes, and the zeroing of intra-cluster communications. Nodes in the same cluster have similar fill patterns and colors.

Figure 3 shows the schedule map of the CCA algorithm for the sample workflow. The boxes designate the tasks and the numbers inside the boxes denote the task numbers. Tasks with higher execution times are denoted by longer boxes. Therefore, the execution start and finish time of each task can be specified by the beginning and the end time of each box according to the time bar underneath each step. The *start* and *end* nodes with zero computation and communication time have no effect on the scheduling procedure and are not shown in the schedule map. The example's four scheduling steps are as follows:

- The first step, which is described in more detail in Fig. 3, maps each cluster to a single-core resource for execution. Consequently, four single-core processors, depicted as S1, S2, S3 and S4 in the figure, are leased for 2, 3, 4, and 2 leasing time slots, respectively. In this case, the workflow makespan is 49 min, and the total execution cost is $1.1. Therefore, in this step, the workflow does not meet the specified deadline.
- The second step attempts to reduce the makespan so that the schedule map meets the deadline. As expected in this stage, the scheduling method reduces the makespan which then increases the execution cost. This stage combines the cluster consisting of nodes 0 and 2 with the cluster composed of nodes 1, 4, and 6 and maps them on a dual core resource. Hence, two single-core processors, namely S3 and S4, remain unchanged and one dual core processing resource, D1, is leased. This combination reduces the makespan to 46 min so that the deadline is met. The execution cost in this stage rises to $1.3.

– With the deadline being met in the previous stage, the next step reduces the execution cost of the workflow while taking care not to pass the defined deadline. Therefore, the cluster consisting of tasks 3, 5, and 8, which are executed on a S3 single-core processor, is combined with the cluster composed of tasks 0, 1, 2, 4, and 6, which are executed on a dual core resource. Figure 3 clearly shows that this combination reduces the free time gaps between the tasks in the schedule map. In addition, mapping these tasks for execution on one resource substantially reduces the data communications between these tasks since the bandwidth inside a resource is very high. The data communication between tasks on the same resource is approximately zero. Consequently, the single-core processor is leased for two leasing periods and the dual core resource is leased for four time slots. The makespan in this stage is 46 min, and the execution cost is reduced to $1.

– The last step combines the cluster consisting of tasks 7 and 9 with the tasks executing on the dual core processor. Therefore, in this stage, all the workflow tasks are executed on a dual core resource for 5 leasing time slots. This combination reduces the makespan to 43 min, and the execution cost remains $1.

As predicted, foremost, the scoring method can reduce the makespan to meet the deadline. After the deadline is met, the scoring function reduces the execution cost of the workflow. This reduction in the makespan and the execution cost is mainly achieved by reducing the free time gaps in the schedule map, which also increases the utilization of the leased resources. Moreover, mapping tasks onto the same resource zeros intra-cluster communications positively impacting the reduction of free time gaps and increasing the utilization of resources. In the next section, Fig. 5 shows in more detail the effect of the proposed scheduling algorithm on the makespan and execution cost in a larger scientific workflow.

4.5 Computational complexity

To analyze the computational complexity of the CCA algorithm, consider m as the total number of workflow tasks, n as the number of primary clusters, and r as the total number of available cores of the processing resources. The main part of the CCA algorithm starts on Line 9 of Algorithm 1. Since the instructions inside the loop will be repeated $(n - 2)$ times, therefore, the complexity of this algorithm will be $t * (n - 2)$. Variable t denotes the computational complexity of each iteration of the loop. To compute t, the complexity of the *Calculate_Combination* function must be analyzed. The time complexity of the main loop of Algorithm 2 in Line 5 is $O(p^2)$, where p is the number of clusters in the *Cluster_Subset* which in the worst-case scenario, no combination is fulfilled within the subset and will be $O(n^2)$. On the other hand, this loop tests the serial combination and the parallel combination of the cluster pairs which is equal to the complexity of the HEFT algorithm $O(mr)$. Therefore, the complexity of the *Calculate_Combination* function is $O(n^2mr)$. In the worst-case scenario this function will be called n times; therefore, the overall complexity of the proposed algorithm is $O(n^3mr)$. However, in practical cases, this time complexity is very rarely reached because of the conditions that need to be fulfilled for the worst-case scenario. Moreover, the proposed pruning technique that only computes the score of

cluster pairs that have a time overlap improves the actual execution of the proposed algorithm.

5 Performance evaluation

In the following section, simulation results and the performance evaluation of the proposed method is presented.

5.1 Experimental setup

For the evaluation process, well-known scientific workflows are employed which are benchmarks in research [4,44]. These graphs are based on real scientific workflows in different fields of science such as geology, astronomy, physics, and genetics, and which have different sizes from the aspect of the number of nodes: Montage, SIPHT, Epigenomics, Cybershake, and LIGO. Figure 4 provides the approximate structure of these workflows which have a small number of nodes. The workflows also include features, such as data aggregation, pipelining, data distribution, and redistribution

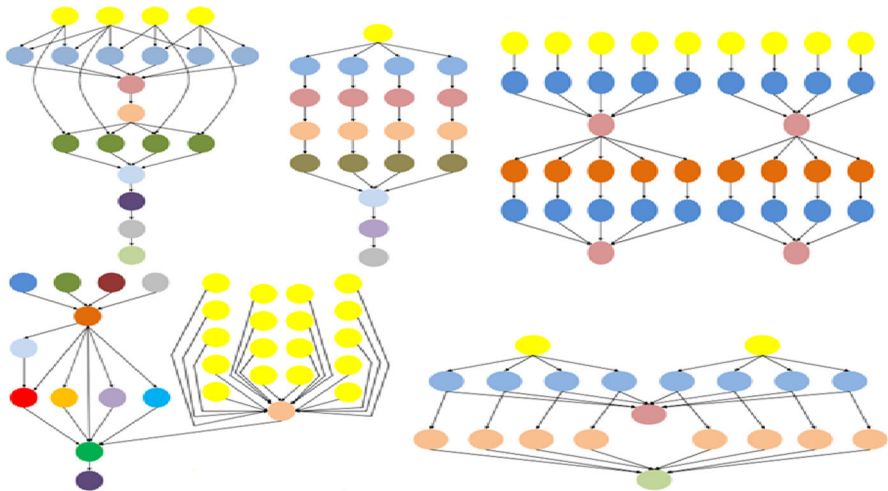| Table 1 Specifications of processing instances and leasing prices		Vcpu	Memory (GiB)	Windows usage (per hour)
	m3.medium	1	3.75	$0.067
	m3.large	2	7.5	$0.133
	m3.Xlarge	4	15	$0.266
	m3.2Xlarge	8	30	$0.532

Fig. 4 Scientific workflow DAGs (*top row from left* Montage, Epigenomics, LIGO. *Bottom row from left* SIPHT, CyberShake) [43]

[41]. Table 1 presents the Amazon EC2 instances considered in this research and their related leasing prices (https://aws.amazon.com/ec2/pricing/).

The proposed algorithm has been compared with one of the most cited algorithms in this context, the hybrid cloud optimized cost (HCOC) algorithm, proposed by Bittencourt et al. [2]. This method schedules the workflow on a hybrid cloud and, most importantly, utilizes multicore processing resources. The main goal of this algorithm is to minimize the monetary costs of the workflow execution with respect to the specified deadline constraint, objectives very similar to the present work's proposed method. The HCOC receives an initial schedule from the PCH method and tries to reduce the costs and also meet the deadline by iteratively rescheduling the workflow tasks on processing resources.

To compare the current study's simulation results with those of the HCOC algorithm, first a reasonable deadline for each workflow must be defined. Therefore, a method which defines five different possible deadlines for each workflow is employed, as follows:

$$Deadline = M_f + \alpha \ (M_s - M_f) \tag{13}$$

where M_s is the workflow makespan when it consists of only the primary clusters and each cluster is running on a single-core processing resource. In other words, M_s denotes the makespan of the longest schedule. Also, M_f denotes the makespan when all available clusters have been combined into a single cluster, and all communication costs are zero. Similarly, M_f is the makespan of the fastest schedule and α is the *deadline factor* which can be set between 0 and 1. If the *deadline factor* is set to 0, the *Deadline* will be M_f which is the fastest makespan. In practice, however, meeting such a deadline is, in fact, impossible. Likewise, setting the *deadline factor* to 1 will result in making M_s the deadline. Hence, setting the *deadline factor* in the range between 0.5 and 0.9 gradually increased the deadline in the current research's tests.

Figure 5 shows the scheduling procedure of the proposed algorithm and the HCOC method from the makespan perspective on the Montage workflow with 50 nodes and the deadline factor set to 0.7. In this figure, the unit of bandwidth is in bytes and the average makespan in minutes. What is first noted is that, as the bandwidth gradually increases, the deadline steadily decreases. To explain, by augmenting the bandwidth, the data transmission time between nodes decreases, thus directly reducing the workflow makespan. Figure 5 demonstrates that the proposed method selects resources in such a way that the workflow makespan is very close to the defined deadline and yet does not pass it so as to decrease execution costs, since we have assumed that reducing the workflow makespan has no advantage for the user. In other words, the CCA algorithm attempts to use available free time gaps in the schedule map instead of leasing new resources. In contrast, the HCOC method employs the HEFT scheduler which usually selects solutions that do not consider execution costs and attempts to reduce the overall makespan with respect to the available resources. Furthermore, increasing the bandwidth reduces the data communications between clusters mapped on separate resources. Therefore, there is a reduction in the difference between mapping clusters with high communications on the same multicore resources and executing them on separate resources and so the outcome of the two methods is similar in higher bandwidths.
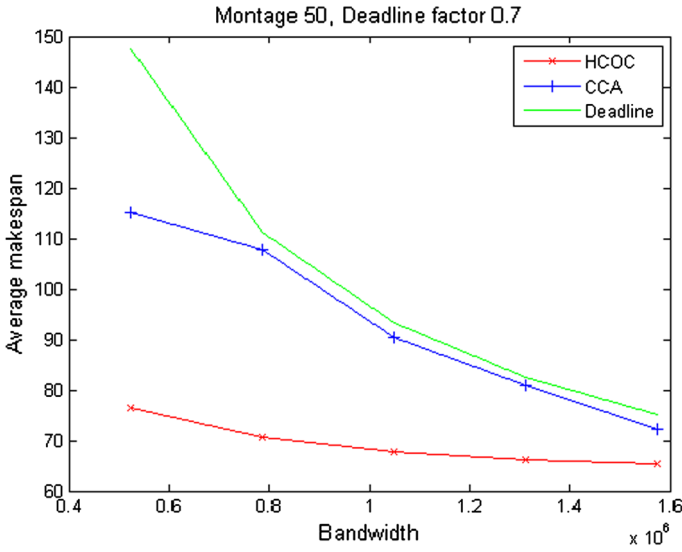
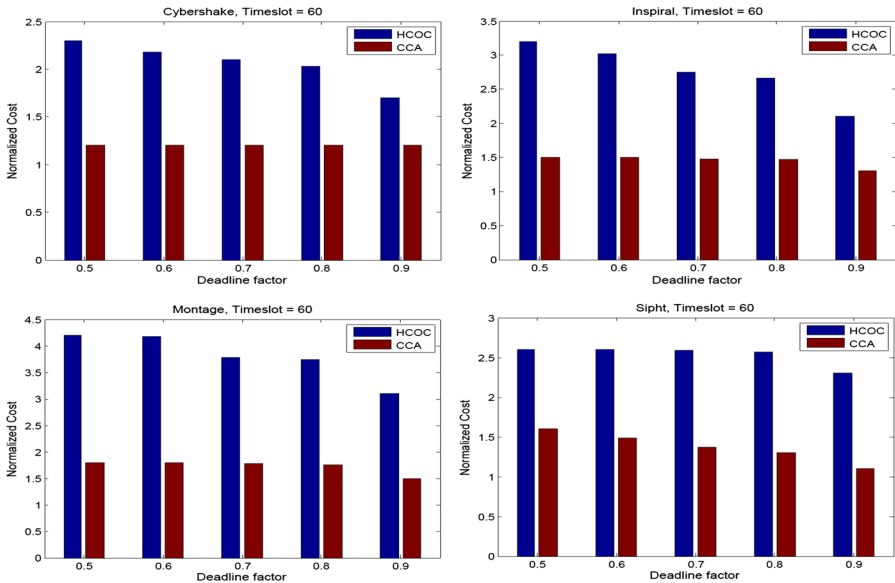**Fig. 5** Makespan results on the Montage workflow

Fig. 6 Normalized cost of scheduling using CCA and HCOC on small workflows

Figure 6 shows the execution costs of the CCA and HCOC algorithm on small workflows consisting of about 25–30 nodes with 60-min leasing time slots. A random number between 0.95 and 1.05 was generated, and this random number was multiplied with the execution and communication weights to create one hundred different workflows of the same kind, but with a very small variation in computations and

communications for each step. The results show the average of these one hundred workflows for each case. Since the attributes of the workflows may differ the cost must be normalized to make the comparison easier. Therefore, the normalized cost in these charts was considered. This is computed by dividing the current execution cost of the workflow by the execution cost of the cheapest possible schedule. The charts in Fig. 6 demonstrate that the CCA method outperforms the HCOC algorithm in all cases. One of the most impressive characteristics of the proposed method is that it works very well for communication-intensive workflows; by scheduling multiple clusters on a multicore processor, the communication between these clusters is reduced to zero, especially if resources with a higher number of cores are used. Thus, processing resources with more cores result in increased parallel execution of workflow clusters. In addition, mapping more clusters onto the same resource reduces the communication between these clusters down to zero, which significantly impacts the makespan. For example, in the Montage workflow, the CCA cost results are much lower than those of the HCOC method. The goal of this scheduling method is to choose processing resources in such a way that the workflow makespan is closest to the defined deadline. This method allows the scheduling algorithm to select resources which result in the minimization of free time gaps. Minimizing free time gaps overall reduces the needed leased resource time periods, maximizes the utilization of the leased resources, and so directly lowers the execution cost. Therefore, in cases in which the deadline of the workflow is not strict, the schedule map changes to reduce execution costs with the constraint of not passing the specified deadline. This situation is clearly presented in Fig. 6. In cases in which the defined deadline is tight (e.g., deadline factor= 0.5), the proposed scheduling method chooses processing instances so that the deadline can be met. This leads to more free time gaps in the schedule map and leased processing instances, which raises the cost in tight deadlines. However, increasing the deadline factor removes the tight constraint, and the execution costs can be lowered by the algorithm. In other words, if the workflow has no deadline constraint, then a single resource may be used for executing the workflow. The workflow tasks can be mapped so that the tasks' precedence is observed and the free time gaps also minimized. In this case, the execution cost of the workflow is minimized. Generally, the CCA method attempts to reduce the data communications between clusters.

Figure 7 shows the same experiment for the same workflows but with approximately 100 node sizes. The resulting pattern is very close to that of experiments conducted with small workflows. The current research employed the same random generation procedure to create one hundred different workflows of the same type, and the results are the average normalized costs of these workflows. As the number of tasks in each workflow rises, the chances increase of combining clusters that can be executed in parallel. Therefore, these clusters can be mapped on resources with a higher number of cores for execution. This procedure leads to zeroing more data communications between the clusters. Furthermore, more tasks are available to fill the existing free time gaps in the schedule map. Hence, increasing the number of tasks in each workflow has a positive effect on the normalized cost of the workflow. This can be clearly seen in Fig. 7's Inspiral workflow, in which the normalized cost lowered by the increase in the number of tasks from 30 to 100. On the other hand, the structure of the Montage workflow shows that this workflow is made up of small clusters with low time overlaps,
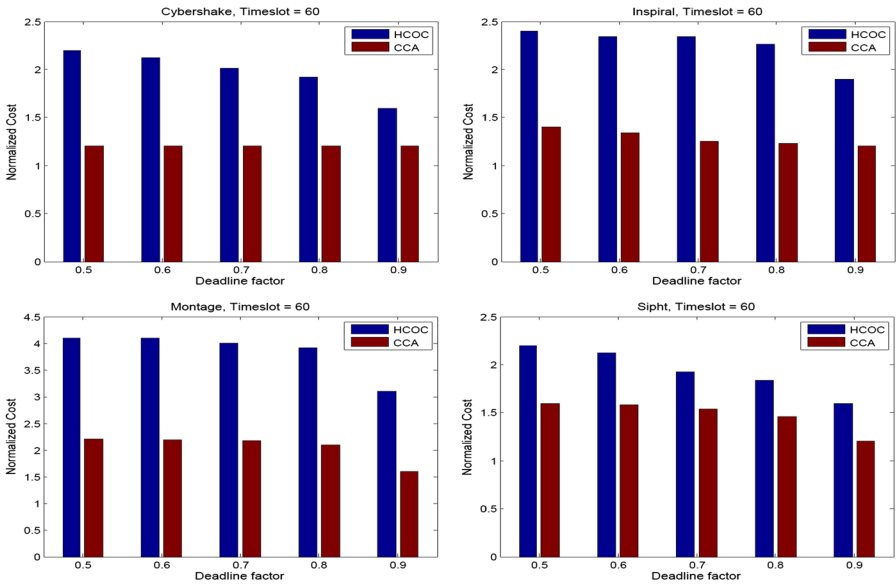
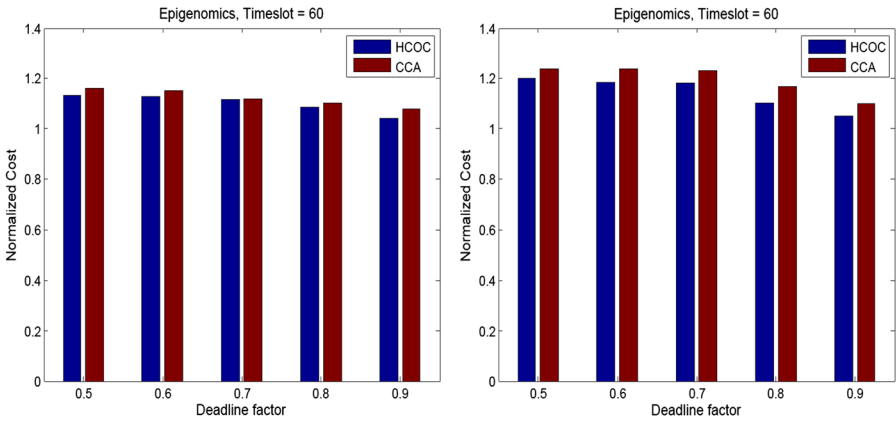**Fig. 7** Normalized cost of scheduling using CCA and HCOC on large workflows

Fig. 8 Normalized cost of scheduling using CCA and HCOC on a small and large Epigenomics workflow (*left* small Epigenomics, *right* large Epigenomics)

which cannot be executed in parallel thus increasing free time gaps in the resource scheduling. Therefore, increasing the tasks of this workflow raises the normalized cost.

6 Discussion

The current study reports that the proposed algorithm shows superior results in communication-intensive workflows, mainly because this method attempts to com-

bine clusters with high data communications. Therefore, in computation-intensive cases with very little communication between tasks, the proposed algorithm might not perform as expected.

Figure 8 presents the results for Epigenomics workflows with 24 and 100 tasks, respectively. In an Epigenomics workflow, the HCOC method performs better in all five deadline factor cases. The currently proposed method does not perform well in the Epigenomics case, because in this workflow, there is a very limited interaction between the selected clusters of the PCH algorithm. Since the present work's method tries to combine clusters with high data communications, combining primary clusters improves the leasing costs. On the other hand, the HCOC method removes the cluster structure task by task and allows tasks to be scheduled by the HEFT algorithm, which increases the chances of finding a completely different and better schedule map.

7 Conclusion and future works

The present paper proposes a new workflow scheduling algorithm on the IaaS cloud which makes use of available multicore processing resources. The main goal of the proposed method is to reduce monetary costs while not passing the user-defined deadline. The main difference between the proposed algorithm and previous similar studies is that the present work utilizes a flexible scoring approach to combine the available clusters in the workflow. This scoring considers different criteria when combining clusters, such as leasing cost, makespan, and resource utilization. The scoring function is adjusted in such a way that the cluster combinations reduce workflow costs while not passing the user-defined deadline. In cases where the workflow makespan does not meet the deadline, the current method attempts to reduce the makespan by leasing processing resources with a higher number of cores and so undertakes larger free time gaps in the schedule map. Therefore, the algorithm raises the monetary costs until the deadline is reached. After meeting the deadline, the present method strives to lower execution costs by filling the free time gaps with tasks from other clusters by combining them. The proposed method was evaluated by comparing the monetary costs of running the workflows with the HCOC method. The experimental results indicate that the currently proposed method outperforms the HCOC algorithm in almost all cases. In the future, the present study's algorithm shall be extended so that it also performs well with computation-intensive workflows with small data communications between tasks. The proposed algorithm will also be enhanced on the real IaaS cloud platform, which can tolerate inaccurate task computation estimations and data communications between the tasks.

References

1. Abrishami S, Naghibzadeh M, Epema DHJ (2012) Cost-driven scheduling of grid workflows using partial critical paths. Parallel Distrib Syst IEEE Trans 23(8):1400–1414
2. Bittencourt LF, Madeira ERM (2011) HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds. J Internet Serv Appl 2(3):207–227
3. Naghibzadeh M (2016) Modeling and scheduling hybrid workflows of tasks and task interaction graphs on the cloud. Future Gen Comput Syst 65. doi:10.1016/j.future.2016.05.029

4. Deelman E, Singh G, Su M-H, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J (2005) Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci Progr 13(3):219–237

5. Berman F, Casanova H, Chien A, Cooper K, Dail H, Dasgupta A, Deng W, Dongarra J, Johnsson L, Kennedy K (2005) New grid scheduling and rescheduling methods in the GrADS project. Int J Parallel Progr 33(2–3):209–229

6. Wieczorek M, Prodan R, Fahringer T (2005) Scheduling of scientific workflows in the ASKALON grid environment. ACM SIGMOD Rec 34(3):56–62

7. Abrishami S, Naghibzadeh M, Epema DHJ (2013) Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gen Comput Syst 29(1):158–169

8. Abraham A, Buyya R, Nath B (2000) Nature's heuristics for scheduling jobs on computational grids. In: The 8th IEEE international conference on advanced computing and communications (ADCOM 2000), pp 45–52

9. Aggarwal AK, Kent RD (2005) An adaptive generalized scheduler for grid applications. In: 19th International symposium on, high performance computing systems and applications, HPCS 2005, pp 188–194

10. Chang V (2014) An introductory approach to risk visualization as a service. Open J Cloud Comput 1:1–9

11. Chang V (2014) The business intelligence as a service in the cloud. Future Gen Comput Syst 37:512–534

12. Arabnejad H, Barbosa JG (2014) A budget constrained scheduling algorithm for workflow applications. J Grid Comput 12(4):665–679

13. Dougan A, Özgüner F (2005) Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. Comput J 48(3):300–314

14. Singh G, Kesselman C, Deelman E (2007) A provisioning model and its comparison with best-effort for performance-cost optimization in grids. In: Proceedings of the 16th international symposium on high performance distributed computing, pp 117–126

15. 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):177–188

16. Szabo C, Kroeger T (2012) Evolving multi-objective strategies for task allocation of scientific workflows on public clouds. In: Evolutionary computation (CEC), IEEE congress on, pp 1–8

17. Maheswaran M, Ali S, Siegal HJ, Hensgen D, Freund RF (1999) Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: 8th Proceedings on, heterogeneous computing workshop (HCW'99), pp 30–44

18. Maheswaran M, Ali S, Siegel HJ, Hensgen D, Freund RF (1999) Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J Parallel Distrib Comput 59(2):107–131

19. Wu F, Wu Q, Tan Y (2015) Workflow scheduling in cloud: a survey. J Supercomput 71(9):3373–3418

20. Topcuoglu H, Hariri S, Wu M (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. Parallel Distrib Syst IEEE Trans 13(3):260–274

21. Sarkar V (1989) Partitioning and scheduling parallel programs for multiprocessors. Pitman, MA, USA Cambridge

22. Bittencourt LF, Madeira ERM (2008) A performance-oriented adaptive scheduler for dependent tasks on grids. Concurr Comput Pract Exp 20(9):1029–1049

23. Yao G, Ding Y, Jin Y, Hao K (2016) Endocrine-based coevolutionary multi-swarm for multi-objective workflow scheduling in a cloud system. Soft Comput. doi:10.1007/s00500-016-2063-8

24. Grandinetti L, Pisacane O, Sheikhalishahi M (2013) An approximate ϵ-constraint method for a multi-objective job scheduling in the cloud. Future Gen Comput Syst 29(8):1901–1908

25. Chang W-L, Ren T-T, Feng M (2015) Quantum algorithms and mathematical formulations of biomolecular solutions of the vertex cover problem in the finite-dimensional hilbert space. NanoBiosci IEEE Trans 14(1):121–128

26. Aggarwal M, Kent RD, Ngom A (2005) Genetic algorithm based scheduler for computational grids. In: 19th International symposium on, high performance computing systems and applications, HPCS 2005, pp 209–215

27. Alhusaini AH, Prasanna VK, Raghavendra CS (1999) A unified resource scheduling framework for heterogeneous computing environments. In: 8th Proceedings on, heterogeneous computing workshop (HCW'99), pp 156–165

28. Bajaj R, Agrawal DP (2004) Improving scheduling of tasks in a heterogeneous environment. Parallel Distrib Syst IEEE Trans 15(2):107–118
29. Yu J, Ramamohanarao K, Buyya R (2009) Deadline/budget-based scheduling of workflows on utility grids. In: Buyya R, Bubendorfer K (eds) Market-oriented grid and utility computing. Wiley, Hoboken, pp 427–450
30. Zheng W, Sakellariou R (2013) Budget-deadline constrained workflow planning for admission control. J Grid Comput 11(4):633–651
31. Yang T, Gerasoulis A (1994) DSC: scheduling parallel tasks on an unbounded number of processors. Parallel Distrib Syst IEEE Trans 5(9):951–967
32. Bittencourt LF, Madeira ERM (2010) Towards the scheduling of multiple workflows on computational grids. J Grid Comput 8(3):419–441
33. Garg SK, Yeo CS, Anandasivam A, Buyya R (2011) Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers. J Parallel Distrib Comput 71(6):732–749
34. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gen Comput Syst 28(5):755–768
35. Younge AJ, Von Laszewski G, Wang L, Lopez-Alarcon S, Carithers W (2010) Efficient resource management for cloud computing environments. In: Green computing conference international, pp 357–364
36. Nathani A, Chaudhary S, Somani G (2012) Policy based resource allocation in IaaS cloud. Future Gen Comput Syst 28(1):94–103
37. Wang W, Zeng G, Tang D, Yao J (2012) Cloud-DLS: dynamic trusted scheduling for cloud computing. Expert Syst Appl 39(3):2321–2329
38. Frîncu ME (2014) Scheduling highly available applications on cloud environments. Future Gen Comput Syst 32:138–153
39. Malawski M, Juve G, Deelman E, Nabrzyski J (2012) Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In: Proceedings of the international conference on high performance computing, storage and analysis, networking, p 22
40. Abrishami S, Naghibzadeh M (2011) Budget constrained scheduling of grid workflows using partial critical paths. In: World congress in computer science, computer engineering, and applied computing (WORLDCOMP'11)
41. Poola D, Garg SK, Buyya R, Yang Y, Ramamohanarao K (2014) Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In: The 28th IEEE international conference on advanced information networking and applications (AINA-2014), pp 1–8
42. Deelman E, Gannon D, Shields M, Taylor I (2009) Workflows and e-Science: an overview of workflow system features and capabilities. Future Gen Comput Syst 25(5):528–540
43. Yu J, Buyya R (2005) A taxonomy of workflow management systems for grid computing. J Grid Comput 3(3–4):171–200
44. Bharathi S, Chervenak A, Deelman E, Mehta G, Su M-H, Vahi K (2008) Characterization of scientific workflows. In: Workflows in support of large-scale science, WORKS 2008, 3 Workshop on, pp 1–10