Cluster Computing (2021) 24:701–715
https://doi.org/10.1007/s10586-020-03151-w

Execution cost minimization scheduling algorithms for deadline-constrained parallel applications on heterogeneous clouds

Weihong Chen[1,2] · Guoqi Xie[1,3] · Renfa Li[1,3] · Keqin Li[4]

Received: 6 July 2017 / Revised: 26 June 2020 / Accepted: 29 June 2020 / Published online: 15 July 2020

Abstract

The problem of minimizing the execution monetary cost of applications on cloud computing platforms has been studied recently, and satisfying the deadline constraint of an application is one of the most important quality of service requirements. Previous method of minimizing the execution monetary cost of deadline-constrained applications was the "upward" approach (i.e., from *exit* to *entry* tasks) rather than combining the "upward" and "downward" approaches. In this study, we propose monetary cost optimization algorithm (DCO/DUCO) by employing "downward" and "upward" approaches together to solve the problem of execution cost minimization. "Downward" cost optimization is implemented by introducing the concept of the variable deadline-span and transferring the deadline of an application to each task. On the basis of DCO, the slack time is utilized to implement "upward" cost optimization without violating the precedence constraints among tasks and the deadline constraint of the application. Experimental results illustrate that the proposed approach is more effective than the existing method under various conditions.

Keywords Cost optimization · DAG scheduling · Deadline constraint · Heterogeneous clouds · Workflows

1 Introduction

1.1 Background

Cloud computing has become one of the most attractive platforms that can provide consumers a cost-efficient computing service to execute various workflows [1, 2]. Large-scale science workflow applications, such as traffic prediction and e-commerce, are often represented by directed acyclic graphs (DAGs) [3, 4]. At present, a large-scale scientific workflow application can at least contain hundreds of tasks and has high performance computing requirements, such as reasonable time and low cost. Cloud computing can provide consumers cost-efficient computing services through the service level agreement (SLA) that defines the quality of service (QoS) on the basis of the pay-as-you-go cost model [5]. In order to speed up the processing, some tasks can be executed in parallel in the cloud computing system, which contains heterogeneous resources with various computing capabilities and prices [6, 7].

✉ Guoqi Xie
xgqman@hnu.edu.cn

Weihong Chen
whchen@hnu.edu.cn

Renfa Li
lirenfa@hnu.edu.cn

Keqin Li
lik@newpaltz.edu

[1] College of Computer Science and Electronic Engineering, Hunan University, Changsha, China

[2] College of Information and Electronic Engineering, Hunan City University, Yiyang, China

[3] Key Laboratory for Embedded and Network Computing of Hunan Province, Changsha, China

[4] Department of Computer Science, State University of New York, New Paltz, New York 12561, USA

1.2 Motivation

DAG scheduling is the problem of mapping each task of the application to a suitable processor to satisfy a specific performance criterion [8]. The tradeoff between time and cost of executing a deadline-constrained parallel application is the dilemma in solving the DAG scheduling problem. The scheduling problem becomes a challenge on current and future heterogeneous cloud platforms. In a cloud environment, service providers and customers are the two parties with conflicting SLA requirements [5]. Minimizing the execution cost of the application without violating the SLA is one of the most important concerns of service providers. For customers, the deadline constraint of the application is an important property of QoS requirements.

Several studies have been conducted to optimize DAG scheduling under grid and cloud computing environments [9–14]. These studies have raised different approaches for different scheduling targets, such as makespan minimization, cost minimization, throughput maximization, and resource utilization optimization, to solve QoS-required DAG scheduling. In addition, the DAG scheduling of deadline-constrained applications in cloud computing environments has attracted the attention of researchers [15–19]. The main idea of these algorithm is to transfer the deadline constraint of an application to that of each task and ensure that each task can be completed within individual sub-deadline constraint. Abrishami et al. presented the IaaS cloud partial critical paths (IC-PCP) algorithm, in which tasks can be assigned to the admissible resource with minimum cost based on the latest finish time (LFT) while satisfying the deadline constraint of the application [15]. Although the IC-PCP algorithm significantly improved the performance of cost optimization for deadline-constrained parallel applications, it has its limitation: the IC-PCP algorithm optimizes the execution cost of tasks only from the upward perspective (i.e., from the exit task to the entry task) as the *LFT* constraint of each task is being satisfied. The deadline span that is equal to the difference between the latest finish time and the earliest finish time of the application is used in the *LFT*. The IC-PCP algorithm transfers the deadline constraint of the application to each task of the application with the same deadline span, which can easily lead to violate the deadline constraint of the application. However, the variable deadline-span is used in our study, and the upward and downward cost optimization approaches are proposed to satisfy the deadline constraint of the application.

1.3 Contributions of the study

In this study, the objective is to minimize the total execution cost of parallel applications under the deadline constraint. The DCO/DUCO algorithm using a variable deadline-span is proposed for deadline-constrained parallel applications. The contributions of this study are as following:

(1) The algorithm of downward cost optimization (DCO) is proposed. "Downward" means that cost minimization is implemented from the entry task to the exit task according to the non-increasing order of rank upward ($rank_u$) values. DCO transfers the deadline constraint of the application to sub-deadline of each task, and minimizes the total cost by defining a variable deadline-span without violating the deadline constraint of the application.

(2) The algorithm of downward-upward cost optimization (DUCO) is presented as a supplement to further reduce the execution cost of the application. "Upward" means that the cost minimization is implemented from the exit task to the entry task according to the non-decreasing order of the $rank_u$ values. DUCO can eliminate or reduce the slacks between adjacent tasks in the same processor as the deadline constraint of the application is being satisfied.

(3) Simulated experiments with parallel applications are conducted under different deadline-constrained and scale conditions to verify that the proposed DUCO can obtain the minimum execution cost compared with the state-of-the-art algorithm.

The rest of this study is organized as follows. Section 2 reviews related literature. Section 3 presents the models and problem formulation. Section 4 presents the preliminaries and DCO algorithm. Section 5 presents the DUCO algorithm. Section 6 evaluates the verification methods with experiments. Section 7 concludes this study.

2 Related works

The DAG scheduling problem has been studied extensively and various heuristic approaches have been proposed by numerous researchers because of the demands for high-performance computing for large-scale workflow applications [20–24]. These heuristics are classified into a variety of categories, such as single and multiple QoS parameters scheduling algorithms [25–27]. Time and execution cost are the common parameters considered in scheduling strategies [28–32]. The schedule length, also called makespan, is the major concern for high performance requirements. Topcuoglu et al. proposed the heterogeneous earliest finish time (HEFT) algorithm for heterogeneous systems [6]. HEFT is one of the most popular high-performance scheduling algorithm and plays a role in cost-aware scheduling. The execution cost is an important parameter for consumers in cloud systems. Abrishami et al. [12] proposed a partial critical path algorithm on utility grids to minimize the total execution cost. Convolbo et al. [30] proposed a heuristic DAG scheduling algorithm to optimize the execution cost as the load balancing. The proposed method is hierarchical scheduling in heterogeneous cloud environments, but the communication time between two adjacent tasks is assumed to be zero in their model.

For the optimization scheduling of an application with QoS awareness, one of the dual problems with our study is to minimize the schedule length of the budget-constrained parallel application. Wu et al. [25] proposed a critical-greedy (CG) approach in homogeneous cloud environments to minimize the schedule length for executing the scientific workflow application with the budget constraint. Chen et al. [28] proposed a scheduling strategy called minimizing the schedule length using the budget level (MSLBL) to optimize the schedule length for budget-constrained applications. Mao et al. [16] proposed the auto-scaling computational instances approach to optimize the cost of deadline-constrained parallel applications in cloud environments. Rodriguez et al. [17] proposed an approach of particle swarm optimization (PSO) to optimize the number of computational instances for deadline-constrained parallel application. The PSO has a high time complexity because the initial configuration and parameter training is time-consuming. Reasonable results can be obtained in a relatively short period of time only if the PSO parameters are properly tuned in advance. Abrishami et al. [15] designed a QoS-based workflow scheduling algorithm called IaaS cloud partial critical paths for cloud environments, which aims to minimize the execution cost for deadline-constrained parallel applications. IC-PCP distributed the overall deadline of the workflow across individual tasks, and the task on the critical path was first scheduled to minimize the cost before their individual sub-deadline. Compared with our studies, their model corresponds to the homogeneous cloud environment with an unbounded set of resources. Moreover, the number of computational instances is increased in the demand-supply mode.

In this study, we focus on the cost optimization problem for the DAG scheduling of deadline-constrained parallel applications in heterogeneous cloud environments, in which the communication time between tasks and the execution cost on heterogeneous processors are considered, and the number of processors is bounded.

3 System models and problem formulation

In this section, we describe an application model, a cost model and problem formulation, which form the basis of our approach. Table 1 introduces the important notations and their definitions as used in this study.

3.1 Application model

The objective of this study is to minimize the execution costs of applications by searching for an appropriate allocation decision of mapping tasks into processors on heterogeneous cloud systems. The targeted cloud system consists of a set of heterogeneous processors that provide computing services with different capabilities and costs [10]. Assume that the processor set is $P = \{p_1, p_2, ..., p_{|P|}\}$, where $|P|$ is the size of set P. For any set X, $|X|$ represents the set size in this study. In a DAG, nodes represent tasks and edges represent dependencies between tasks. $G = \{N, E, C, W\}$ represents the DAG of the precedence-constrained application that runs on processors. N and E are the sets of task nodes and communication edges in G, respectively. $n_i \in N$ is a task with different execution times on different processors, and $e_{i,j} \in E$ is a communication message from task n_i to task n_j. Accordingly, C is the set of communication edges, and $c_{i,j}$ represents the communication time between n_i and n_j if they are not assigned to the

Table 1 Important notations in this study

Notation	Definition
$w_{i,k}$	Execution time of task n_i running on processor p_k
$c_{i,j}$	Communication time between n_i and n_j
$pred(n_i)$	Set of predecessors of task n_i
$succ(n_i)$	Set of successors of task n_i
$rank_u(n_i)$	Upward rank value of task n_i
$f(i)$	Index of the processor assigned to task n_i
$lb(G)$	Lower bound of the application G
$price_k$	Unit price of processor p_k
$cost_{i,k}$	Cost of task n_i on processor p_k,
$cost(G)$	Total cost of the application G
$deadline(n_i)$	Deadline of task n_i
$deadline(G)$	Deadline of the application G
$dspan(n_i)$	Deadline span of task n_i
$dspan(G)$	Deadline span of the application G
$makespan(G)$	Schedule length of the application G
$EST(n_i, p_k)$	Earliest start time of task n_i on processor p_k
$EFT(n_i, p_k)$	Earliest finish time of task n_i on processor p_k
$LFT(n_i, p_k)$	Latest finish time of task n_i on processor p_k
$AFT(n_i)$	Actual finish time of task n_i

same processor. W is an $|N| \times |P|$ matrix, where $w_{i,k}$ is the execution time of task n_i running on processor p_k. $pred(n_i)$ is the set of predecessors of task n_i, and $succ(n_i)$ is the set of successors of task n_i. In a given DAG, the task without a predecessor is the entry task denoted as n_{entry}, whereas the task without any successor is the exit task denoted as n_{exit}. If a DAG has multiple n_{entry} or n_{exit} tasks, then a dummy entry or exit task with zero-weight dependencies is added to the graph.

Figure 1 shows a motivating parallel application with ten tasks [6]. The weight of the edge between n_1 and n_4 represents the communication time, which is denoted by $c_{1,4} = 9$ if n_1 and n_4 are not assigned to the same processor. The predecessor of task n_7 is n_3, which is denoted by $pred(n_7) = \{n_3\}$. Similarly, the successor of task n_2 is n_8 and n_9, which is denoted by $succ(n_2) = \{n_8, n_9\}$.

The tasks in the motivating example are assumed to be executed on three processors $\{p_1, p_2, p_3\}$. Table 2 shows the execution time matrix $|N| \times |P|$ of tasks on different processors of the motivating parallel application. The execution time of task n_1 on processor p_3 is 9, which is denoted as $w_{1,3} = 9$. Due to the heterogeneity of processors, a task on different processors may have different execution time values.

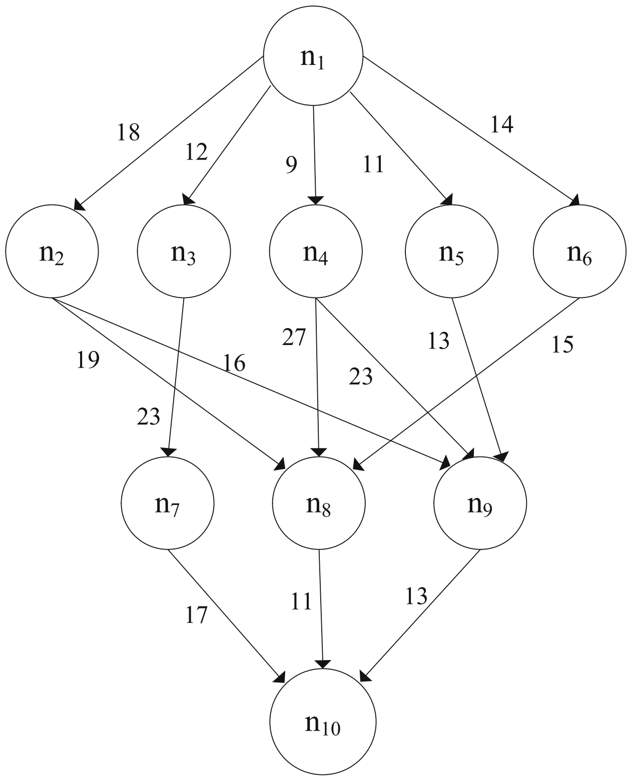


Fig. 1 Motivating example of a DAG-based parallel application with ten tasks [6]

Table 2 Execution time value of tasks on different processors in Fig. 1 [6]

Task	p_1	p_2	p_3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	5	11	14
n_9	18	12	20
n_{10}	21	7	16

3.2 Cost model

The cost model of a DAG based on a pay-as-you-go basis was defined based on the aforementioned application model. Similar to current commercial clouds, the users are charged according to the amount of time that they have used the processors [15]. Each processor has an individual unit price because of the heterogeneity of processors in the system [26, 28].

We assume that $price_k$ is the unit price of processor p_k and computation and storage services on processors are in the same physical region. Therefore, the communication time $c_{i,j}$ is independent of the computation service on the processors and determined by the amount of data to be transferred between task n_i and task n_j, except when tasks n_i and n_j are executed on the same processor, where $c_{i,j} = 0$ [21]. The internal data transfer cost refers to the transfer time of data from task n_i to task n_j executed on the same processor multiplied by the unit price of the processor. Since the data from n_i to n_j needed for computation may be fetched locally, that is, no data transfer is required, the data transfer time of tasks n_i and n_j is zero. Thus, the data transfer cost is assumed to be zero in our model. Accordingly, the cost $cost_{i,k}$ of task n_i on processor p_k and the total cost $cost(G)$ of a DAG are defined as follows:

$$cost_{i,k} = w_{i,k} \times price_k, \tag{1}$$

$$cost(G) = \sum_{i=1}^{|N|} w_{i,f(i)} \times price_{f(i)}, \tag{2}$$

where $f(i)$ is the index of the processor assigned to task n_i, $w_{i,f(i)}$ is the execution time of task n_i on the processor $p_{f(i)}$, and $price_{f(i)}$ is the unit price of the processor $p_{f(i)}$.

We assume that the unit price of heterogeneous processors is known, as shown in Table 3. According to Eqs. (1) and (2), we have the total cost value of executing

Table 3 Unit price of processors employed

p_k	$price_k$
p_1	3
p_2	5
p_3	7

the application in Fig. 1 using the HEFT algorithm is $cost(G) = 612$.

3.3 Problem formulation

The scheduling problem to minimize the execution cost of deadline-constrained applications in heterogeneous cloud environments is constructed, which is called Minimizing Execution Cost - Deadline Constrained (MEC-DC).

Definition 1 (**MEC-DC**). Given a DAG-based parallel application $G = \{N, E, C, W\}$, a set of heterogeneous processors $P = \{p_1, p_2, ..., p_{|P|}\}$ that support different unit prices for executing tasks and a deadline constraint *deadline*(G) to find a task schedule $f : n_i \rightarrow p_k$, $\forall n_i \in N$, $\exists p_k \in P$, such that the total execution cost of the application is minimized without exceeding the schedule length of the deadline constraint, which is formulated as minimize

$$cost(G) = \sum_{i=1}^{|N|} w_{i,f(i)} \times price_{f(i)},$$

subject to

$$makespan(G) \leq deadline(G).$$

Because the MEC-DC scheduling is NP-hard, a heuristic approach is designed in this study to address this optimization problem.

4 Downward cost optimization

The algorithm of DCO, which aims to find a scheduling strategy to minimize the execution cost of deadline-constrained parallel applications in heterogeneous cloud environments, is presented in this section. The key idea of DCO is to transfer the deadline of the application to that of each task.

4.1 Preliminaries

For reducing the schedule length to a minimum with low complexity and high performance in heterogeneous cloud environments, the HEFT algorithm is the most popular DAG scheduling algorithm [12] [13].

Upward rank value. HEFT uses the upward rank value ($rank_u$) of a task as the task priority standard shown in Eq. (3).

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} \{c_{i,j} + rank_u(n_j)\}, \tag{3}$$

where $\overline{w_i}$ is the average execution time of task n_i on processors and is calculated as $\overline{w_i} = \sum_{k=1}^{|P|} w_{i,k}/|P|$. In this case, all tasks in the application are ordered according to the descending order of $rank_u$.

Table 4 shows the $rank_u$ values of tasks of the motivating parallel application (Fig. 1). The task assignment order is $\{n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, n_{10}\}$.

Earliest finish time. $EST(n_j, p_k)$ and $EFT(n_j, p_k)$ represent the earliest start time (EST) and earliest finish time (EFT) of task n_j on processor p_k, respectively. EFT is considered as the task assignment criterion in HEFT because the local optimal of each task can be satisfied. These attributes are calculated by

$$\begin{cases} EST(n_{\text{entry}}, p_k) = 0, \\ EST(n_j, p_k) = max\{T_{avail(p_k)}, \max\limits_{n_i \in pred(n_j)} \{AFT(n_i) + c_{i,j}\}, \end{cases} \tag{4}$$

and

$$EFT(n_j, p_k) = EST(n_j, p_k) + w_{j,k}, \tag{5}$$

where $T_{avail(p_k)}$ is the available *EST* on processor p_k for task execution. $c_{i,j}$ is the actual communication time between n_i and n_j. $AST(n_i)$ is the actual start time (AST) of task n_i, and $AFT(n_i)$ is the actual finish time (AFT) of task n_i. If n_j and its predecessor n_i are assigned to the same resource, then

Table 4 Task assignment of the application in Fig. 1 using HEFT

Task	$rank_u(n_i)$	$AST(n_i)$	$AFT(n_i)$	$p_{f(i)}$
n_1	108	0	9	p_3
n_3	80	9	28	p_3
n_4	80	18	26	p_2
n_2	77	27	40	p_1
n_5	69	28	38	p_3
n_6	63	26	42	p_2
n_9	44	56	68	p_2
n_7	43	38	49	p_3
n_8	36	57	62	p_1
n_{10}	14	73	80	p_2

$c_{i,j} = 0$. The insertion-based scheduling strategy is used for assigning task n_i to the processor with the minimum EFT.

Table 4 shows the assignment of all tasks in the parallel application (Fig. 1) using the HEFT algorithm. The *AST*, *AFT* and *f*(*i*) of each task assignment are provided.

Lower bound and deadline constraints. Similar to state-of-the-art studies [14], this study also takes advantage of the HEFT algorithm to certify the lower bound of a parallel application. When the standard DAG-based scheduling algorithm is used, the minimum schedule length of an application is referred as the lower bound. The lower bound is calculated by

$$lb(G) = \min_{p_k \in P}\{EFT(n_{\text{exit}}, p_k)\}. \tag{6}$$

Then, the relative deadline of an application *deadline*(*G*) is provided for the application, where $deadline(G) \geq lb(G)$. Table 4 shows that *lb*(*G*) is equal to 80.

4.2 Satisfying the deadline constraint

For a given application, *lb*(*G*) has been obtained by using the HEFT algorithm. The deadline-span between the deadline and the lower bound of the application *dspan*(*G*) can be computed as

$$dspan(G) = deadline(G) - lb(G). \tag{7}$$

When the value of *lb*(*G*) is equal to that of *makespan*(*G*), *dspan*(*G*) is also called the slack time of G, denoted as *slack*(*G*).

An awkward way to satisfy the deadline constraint of the application is to set different deadline spans for different tasks and satisfy the following:

$$\sum_{i=1}^{|N|} dspan(n_i) \leq dspan(G), \tag{8}$$

where $dspan(n_i)$ is the deadline span of task n_i.

According to Eq. (8), we have

$$0 \leq dspan(n_i) \leq dspan(G). \tag{9}$$

However, finding the optimal deadline-span for each task by excluding all possible combinations for an application is time consuming. For the application in Fig. 1 with $deadline(G) = 90$, the deadline span of each task is in the scope of [0, 10], and the exhausting number of deadline combinations has reached 11^{10} when only the integer step-size of the deadline span is considered.

For convenient description, the set of task assignments is assumed to be $\{n_{s(1)}, n_{s(2)}, \ldots, n_{s(j)}, n_{s(j+1)}, \ldots, n_{s(|N|)}\}$. Such task set includes three parts, namely, the tasks that have been assigned $\{n_{s(1)}, n_{s(2)}, \ldots, n_{s(j-1)}\}$, the task to be assigned $n_{s(j)}$, and the tasks that have not been assigned $\{n_{s(j+1)}, n_{s(j+2)}, \ldots, n_{s(|N|)}\}$. Initially, the sequence number of the task to be assigned is the first and $j = 1$.

$AFT_{\text{HEFT}}(n_i)$ is the actual finish time of task by using the HEFT algorithm, which is equal to $lb(n_i)$. The deadline constraint of tasks uses the variable deadline span (VDS) to ensure that the deadline constraint of the application can be satisfied at each task assignment. For the variable deadline-span of tasks *vdspan* is between 0 and *dspan*(*G*), so that $vdspan \in [0, dspan(G)]$.

Initially, all tasks of the application are unassigned, and $vdspan = dspan(G)$. Then, when assigning $n_{s(j)}$,

$$deadline(n_{s(j)}) = AFT_{\text{HEFT}}(n_{s(j)}) + vdspan. \tag{10}$$

If task $n_{s(j)}$ can be assigned within $deadline(n_{s(j)})$, a new AFT of task $n_{s(j)}$, $AFT_{\text{DCO}}(n_{s(j)})$, is generated by using the DCO algorithm. The *AFT* change of task $n_{s(j)}$ has an effect on the deadline span of unscheduled tasks in the application. Therefore, *vdspan* can be updated by using Eq. (11) for the next task to be scheduled in the application.

$$vdspan = \min\{dspan(G), \\ dspan(G) - \max_{1 \leq i \leq j}\{AFT_{\text{DCO}}(n_i)\} + \max_{1 \leq i \leq j}\{AFT_{\text{HEFT}}(n_i)\}\}, \tag{11}$$

where $AFT_{\text{DCO}}(n_i)$ and $AFT_{\text{HEFT}}(n_i)$ are the actual finish times of task n_i using the DCO and HEFT algorithms, respectively. If the task $n_{s(j)}$ can not be assigned within $deadline(n_{s(j)})$, the next round operation is launched, in which all tasks of the application are reset as unassigned and *vdspan* is set to a new value according to the scheduling algorithm. The above iterative process for tasks are continued until all tasks have been scheduled.

Thus far, a deadline for each task has been defined. If we can find a proper processor for task n_i that minimizes

$$cost(n_i) = \min_{p_k \in P}\{cost(n_i, p_k)\},$$

subject to

$$EFT(n_i, p_k) \leq deadline(n_i),$$

then the MEC-DC problem of the parallel application is transferred to the problem of each task.

4.3 DCO algorithm

Inspired by the aforementioned formal analysis, we propose the DCO algorithm described in Algorithm 1. The main idea of DCO is to transfer the deadline constraint of the application to that of each task and take the advantage of the variable deadline span to obtain a scheduling with the minimum cost while satisfying the deadline constraint of the application. The core details are explained as follows:

1) In Lines 1–2, *rank*$_\mathrm{u}$, *lb*(G) and *dspan*(G) are calculated.
2) In Lines 3–8, all variable deadline-spans between 0 and *dspan*(G) are traversed until a feasible scheduling with minimum execution cost is obtained. The VDS algorithm is used to find a scheduling of the application with the minimum execution cost as meeting the deadline constraint. It is implemented by two stages: transferring the deadline constraint of the application into sub-deadlines of tasks and minimizing the total cost. The VDS algorithm is invoked iteratively by using *vdspan* to obtain *makespan*(G) and *cost*(G). If *makespan*$(G) \leq$ *deadline*(G), then a feasible scheduling is obtained and we jump out of the *for* loop.

In the VDS algorithm, *deadline*(G) of the application with *vdspan* is initialized, and *deadline*(n_i) is computed using Eq. (10) in Lines 1–5. The processor with minimum cost under satisfying the condition of $EFT(n_i, p_k) \leq deadline(n_i)$ is selected in Lines 6–20. When task n_i is assigned, *vdspan* is updated according to Eq. (11). If all processors cannot satisfy its deadline constraint, then task n_i is assigned to the processor with the minimum EFT.

In terms of time complexity, the DCO algorithm requires the computation of *rank*$_u$, *lb*(G), and *dspan*(G). In the phase of calling the VDS algorithm, the complexity of each task is $O(|N| \times |P|)$ for computing EFT, and $O(|P|)$ for selecting the processor with the minimum EFT. The total time complexity of the DCO algorithm is $O(|N|^2 \times |P| \times V)$, where V is the number of variable deadline spans.

Although the variable deadline span is real-valued in a parallel application, an integer step-size for the application is still provided to improve the searching efficiency.

Algorithm 1 The DCO Algorithm

```
Input: $G = \{N, E, C, W\}$, for $\forall k, p_k \in P, price_k, deadline(G)$
Output: $cost(G)$, $makespan(G)$
1: Compute $rank_\mathrm{u}$ for all tasks, sort the tasks in a list $dlist$ by the non-increasing order of $rank_u$;
2: Compute $lb(G)$ using Eq. (6) and $dspan(G)$ using Eq. (7);
3: for ($vdspan = dspan(G)$; $vspan \geq 0$; $vdspan-$) do
4:     Call the VDS algorithm based on the given $vdspan$, and $makespan(G)$ and $cost(G)$ are
       obtained;
5:     if ($makespan(G) \leq deadline(G)$) then
6:         exit;
7:     end if
8: end for
9: return $cost(G)$, $makespan(G)$.
```

Algorithm 2 The VDS Algorithm

```
Input: $G = \{N, E, C, W\}$, for $\forall k, p_k \in P, price_k, vdspan$
Output: $cost(G)$, $makespan(G)$
1: Sort the tasks in a list $dlist$ by the non-increasing order of $rank_u$;
2: Initialize $cost(G) = 0$, $deadline(G) = lb(G) + vdspan$, $dspan(G) = vdspan$;
3: while (there is a task in $dlist$) do
4:     $n_i = dlist.out$;
5:     $deadline(n_i) = AFT_{\mathrm{HEFT}}(n_i) + vdspan$;
6:     for ($\forall k, p_k \in P$) do
7:         Initialize $cost_{i,k} = \infty$;
8:         Compute $EFT(n_i, p_k)$ using Eq. (5) based on the insertion-based scheduling policy;
9:         if ($EFT(n_i, p_k) \leq deadline(n_i)$) then
10:            $cost_{i,k} = w_{i,k} \times price_k$;
11:        end if
12:    end for
13:    if (no $EFT(n_i, p_k) \leq deadline(n_i)$ exists for all processors) then
14:        Select the processor $f(i)$ with the minimum EFT for task $n_i$;
15:        $cost_{i,f(i)} = w_{i,f(i)} \times price_{f(i)}$;
16:    else
17:        Select the processor $f(i)$ with the minimum cost for task $n_i$;
18:        Update $vdspan$ using Eq. (11);
19:        $vdspan = \min\{dspan(G), dspan(G) - \max_{1 \leq i \leq j}\{AFT_{\mathrm{DCO}}(n_i)\} + \max_{1 \leq i \leq j}\{AFT_{\mathrm{HEFT}}(n_i)\}\}$
20:        $cost(G) = cost(G) + cost_{i,f(i)}$;
21:    end if
22: end while
23: Compute $makespan(G) = AFT(n_{\mathrm{exit}})$;
24: return $cost(G)$, $makespan(G)$.
```

4.4 Example of DCO algorithm

Example 1 We assume that *deadline*$(G) = 90$. Table 5 shows the results in the parallel application in Fig. 1 using the DCO algorithm. Safe scheduling is obtained when *vdspan* is initialized to 6, where *cost*$(G) = 413$ and *makespan*$(G) = 81$. Table 6 lists the task assignment and Fig. 2 shows the task scheduling of the parallel application in Fig. 1 using DCO when *vdspan* $= 6$. Figure 4 shows the scheduling obtained by DCO does not violate the precedence constraints among tasks and the deadline constraint of the application.

5 Downward-upward cost optimization

This section presents a safe downward-upward cost optimization (DUCO) algorithm as a supplement to the cost-efficient scheduling of the parallel application with a deadline constraint. First, *LFT* is defined. Then, the DUCO algorithm described as Algorithm 3 is presented. Lastly, an example using DUDO is provided.

Table 5 Results of the parallel application in Fig. 1 with *deadline*$(G) = 90$ using DCO

deadline(G)	*vdspan*	*makespan*(G)	*cost*(G)	Safe?(Y/N)
90	10	95	479	N
90	9	95	503	N
90	8	95	503	N
90	7	95	503	N
90	6	81	413	Y

Table 6 Task assignment of the application in Fig. 1 with $deadline(G) = 90$ using DCO

n_i	$deadline(n_i)$	$AST(n_i)$	$AFT(n_i)$	$f(n_i)$
n_1	15	0	14	p_1
n_3	33	14	25	p_1
n_4	34	23	31	p_2
n_2	43	25	38	p_1
n_5	43	31	44	p_2
n_6	41	28	37	p_3
n_9	72	54	66	p_2
n_7	55	38	45	p_1
n_8	70	58	63	p_1
n_10	90	74	81	p_2
$cost(G) = 413, makespan(G) = 81 < deadline(G)$				

5.1 Latest finish time

Definition 2 (**LFT**). The *LFT* of tasks is defined as

$$\begin{cases} LFT(n_{\text{exit}}) = deadline(G), \\ LFT(n_i, p_k) = \min\limits_{n_j \in succ(n_i)} \{AFT(n_j) - w_{j,f(j)} - c_{i,j}\}, \end{cases} \tag{12}$$

where $succ(n_i)$ is the set of successors of task n_i, and $AFT(n_j)$ is the actual finish time of task n_j using a scheduling strategy.

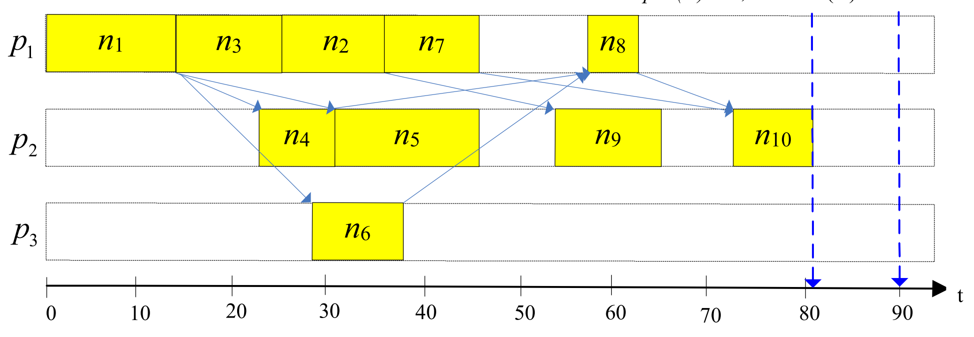
The task scheduling of deadline-constrained applications is considered safe when $makespan(G) \le deadline(G)$ is satisfied. We call the value of $deadline(G) - makespan(G)$ the slack time, denoted as $slack(G)$. The slack time of a safe scheduling using DCO may be greater than zero, which can be utilized to further optimize the execution cost of the application. Correspondingly, $slack(n_i, n_j, p_k)$ indicates the slack time between two adjacent tasks n_i and n_j executed on the same processor p_k. For the exit task, $slack(n_{\text{exit}})$ is equal to the difference between $deadline(G)$ and $makespan(G)$. For example, the $slack(n_7, n_8, p_1)$ in Fig. 2 is 18, and the $slack(n_{\text{exit}})$ is 9.

5.2 DUCO algorithm

On the basis of the aforementioned analysis, the DUCO algorithm described in Algorithm 3 is proposed. The main idea of the DUCO algorithm is that the actual finish time of task n_i may be extended to $LFT(n_i)$ because slacks exist between adjacent tasks on the same processor. Each task selects a cost-efficient processor by the non-decreasing order of $rank_u$ as the precedence constraints among tasks and the deadline constraint of the application are being satisfied. Further details are provided as follows:

1) In Lines 1–3, the parameters for the DUCO algorithm are initialized, i.e., $dspan(G)$.
2) In Lines 4–32, the upward cost optimization is performed on the basis of DCO scheduling only if the value of $dspan(G)$ is larger than zero.
3) In Lines 4–17, check whether the scheduling obtained by using the DCO algorithm can be optimized. If the execution cost of task n_i on processor $f(n_i)$ is not the minimum, the scheduling of task n_i may be further optimized and $flag[n_i] = 1$. When task n_i can be assigned the processor with the lower execution cost within its deadline constraint, the scheduling will be further optimized and $FS = 1$.
4) In Lines 18–32, the processor with lower execution cost is assigned to the task as the slack condition and precedence constraints of tasks are satisfied. If $slack(n_{k1}, n_{k2}, f_{\min}(n_i)) \ge w_{i,f'(n_i)}$ for task n_i with $flag[n_i] = 1$, then a slack on processor $f'(n_i)$ is selected. When the precedence constraints and the deadline constraint of task n_i can be satisfied, $f(n_i)$, $AST(n_i)$ and $AFT(n_i)$ are updated. In Line 25, the *AST* and *AFT* of unassigned tasks in $succ(n_i)$ and the *LFT* of unassigned tasks in $pred(n_i)$ are updated. In Line 27, the execution of task n_i is back translated by a *dspan* time when it can not be optimized.

Fig. 2 Scheduling of the application in Fig. 1 with $deadline(G) = 90$ using DCO



In terms of time complexity, DUCO requires calling the DCO algorithm for parameter initialization that has complexity $O(|N|^2 \times |P| \times V)$. In the upward cost optimization phase, the complexity is $O(|N| \times |P|)$ in searching for tasks to be optimized, and $O(|N^2| \times |P|)$ in performing upward cost optimization. The total complexity of DUCO is $O(|N|^2 \times |P| \times V)$, where |*N*| is the number of tasks, |*P*| is the number of processors, and V is the number of steps in DCO. When the scheduling using DCO is known, the DUCO algorithm has low complexity of $O(|N^2| \times |P|)$ and only needs to perform the upward cost optimization phase.

Algorithm 3 The DUCO Algorithm

```
Input: $G = \{N, E, C, W\}$, for $\forall k, p_k \in P, price_k, deadline(G)$
Output: $cost(G)$, $makespan(G)$
 1: Compute $rank_u$ for all tasks, sort the tasks in a list $uplist$ by the non-decreasing order of
    $rank_u$;
 2: Executing the DCO algorithm and obtain parameters, i.e., $makespan_{\mathrm{DCO}}(G)$, $AST_{\mathrm{DCO}}(n_i)$,
    $AFT_{\mathrm{DCO}}(n_i)$, $f(n_i)$;
 3: Compute $dspan(G) = deadline(G) - makespan_{\mathrm{DCO}}(G)$ ;
 4: if $(dspan(G) > 0)$ then
 5:     $FS = 0$;
 6:     for $(\forall i, n_i \in N)$ do
 7:         flag[$n_i$]=0;
 8:         if ($cost_{i,f(n_i)}$ is not the minimum) then
 9:             flag[$n_i$]=1;
10:             for $(\forall k, p_k \in P)$ do
11:                 Check the execution cost and time of task $n_i$ on $p_k$;
12:                 if (there is a processor with lower execution cost within its deadline of task $n_i$
                    available) then
13:                     $FS = 1$; break;
14:                 end if
15:             end for
16:         end if
17:     end for
18:     if $(FS == 1)$ then
19:         while (there is a task in uplist) do
20:             $n_i = uplist.out$;
21:             Compute $LFT(n_i, p_k)$ for each processor;
22:             if $(flag[n_i] == 1)$ then
23:                 Select an available slack on processor $f_{\min}(n_i)$;
24:                 Update $f(n_i)$, $AST(n_i)$ and $AFT(n_i)$ ;
25:                 Update $AST$ and $AFT$ of unassigned tasks in $succ(n_i)$ and $LFT$ of unassigned
                    tasks in $pred(n_i)$;
26:             else
27:                 $AFT_{\mathrm{DUCO}}(n_i) = AFT_{\mathrm{DCO}}(n_i) + dspan(G)$;
28:             end if
29:         end while
30:     end if
31:     Compute $cost(G)$ using Eq. (2);
32: end if
33: return $cost(G)$ and $makespan(G)$.
```

5.3 Example of DUCO algorithm

Example 2 In this example, $deadline(G) = 90$. Figure 3 shows the scheduling of the parallel application in Fig. 1 using the DUCO algorithm. For example, when DCO is used, n_6 is assigned to p_3, n_7 is assigned to p_1, and $AFT(n_6) = 37$ and $AFT(n_7) = 45$ as shown in Fig. 2. When DUCO is used, n_7 is still assigned to p_1. However, n_6 is switched to p_1, and $AFT(n_6)$ and $AFT(n_7)$ are changed to 51 and 58, respectively, as shown in Fig. 3. Furthermore, the makespan of the parallel application is 82 and the total execution cost is 389. Compared with the scheduling of the parallel application using DCO in Example 1, the execution cost using DUCO is reduced by 5.8% without violating the precedence constraints among tasks and the deadline constraint of the application.

6 Experimental results and discussion

This section shows that the performance comparisons of the DUCO algorithm with DCO, HEFT [6] and IC-PCP [15] algorithms because they have the similar application models. The proposed approach is verified by the simulation method. The simulator that includes workflow applications and the cloud environment modeling is implemented in Java language on a PC platform with Intel Core i5 2.60 GHz CPU and 4 GB memory.

6.1 Experimental workflows

Two parallel workflow applications, namely, fast Fourier transform (FFT) parallel applications and Gaussian elimination parallel applications, are considered. The FFT parallel application is used to characterize low-task parallelism. The Gaussian elimination parallel application is used to characterize high-task parallelism.

The most important information on the workflow application includes the workflow structure, task number, and computational characteristics. The workflow structure, namely, DAG shape, is defined based on five parameters, namely, depth, width, regularity, density, and hops. The number of a DAG is assumed as $|N|$, and the shape parameter is α. The DAG depth is randomly generated from a uniform distribution with a mean value equal to $|N|/\alpha$.

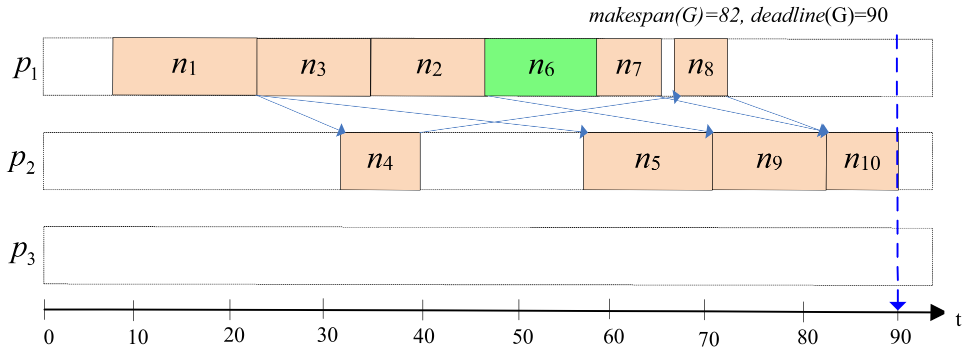


Fig. 3 Scheduling of the application in Fig. 1 with deadline=90 using DUCO

The width for each level is randomly generated from uniform distribution with mean $|N| \cdot \alpha$, which implies a thin DAG with low-task parallelism and a fat DAG with high degree of parallelism. The regularity indicates the uniformity of the number of tasks in each level. The density denotes the number of edges between two DAG levels. A hop is a connection, which indicates an edge that can go from level l to level $l+1$, between two adjacent levels. In our experiment, the parameter ρ is used as the size of the FFT application and its task number is $|N| = 2 \times \rho - 1 + \rho \times \log_2 \rho$, where $\rho = 2^y$ for some integer y. Figure 4 shows an example of the FFT parallel application with $\rho = 8$. ρ exit tasks exist in the FFT application with size ρ. A dummy exit task with zero execution time, which connects these ρ exit tasks with zero communication time, is created to adapt the application of this study. For the Gaussian elimination parallel application, a parameter ρ is used as its matrix size, and the total number of tasks is $|N| = \frac{\rho^2+\rho-2}{2}$ [5]. Figure 5 shows an example of the Gaussian elimination parallel application with $\rho = 5$.

6.2 Experimental setup

For our experiments, we assume that the cloud environment consists of 128 heterogeneous processors with different computing abilities and unit prices, in which the types and the prices of processors are based on the Amazon EC2 environment [2]. The application and processor parameters are: \$ 0.01/h $\leq price_k \leq$ \$ 1/h, 0.01 h $\leq w_{i,k} \leq$ 128 h, 0.01 h $\leq c_{i,j} \leq$ 30 h.

The normalized execution cost NC and the final schedule length $makespan(G)$ of the application are selected as the performance metrics. The NC is expressed by Eq. (13) as follows:

$$NC = cost(G)/cost_{\mathrm{HEFT}}(G), \tag{13}$$

where $cost_{\mathrm{HEFT}}(G)$ is the execution cost of the application using the HEFT algorithm [6]. The scheduling of the application described in Fig. 1 and Table 2 are taken as examples, in which $cost_{\mathrm{HEFT}}(G)$ is 612 and NC is 1.0.

6.3 Experimental results

Two types of parallel applications with different scales and deadline constraints are used to verify the proposed method.

Table 7 and Fig. 6 show the actual makespan and NC of scheduling FFT applications with varying deadline constraints using the HEFT, IC-PCP, DCO, and DUCO algorithms. The size of the application is limited to $\rho = 48$ (i.e., $|N| = 1152$). $deadline(G)$ is changed from $lb(G) \times 1.0$ to $lb(G) \times 1.4$. The makespans obtained by all algorithms are within the required deadline, that is, the four algorithms can satisfy the deadline constraints from the "downward" or "upward" perspectives. The total cost of the application using HEFT is 10960 and the value of NC is 1.0. When the deadline of an application with a certain number of tasks is increased from $lb(G) \times 1.0$ to $lb(G) \times 1.4$, the total costs obtained by IC-PCP, DCO, and DUCO algorithms are decreased. The NC using DCO fluctuates with respect to the total cost obtained by IC-PCP.

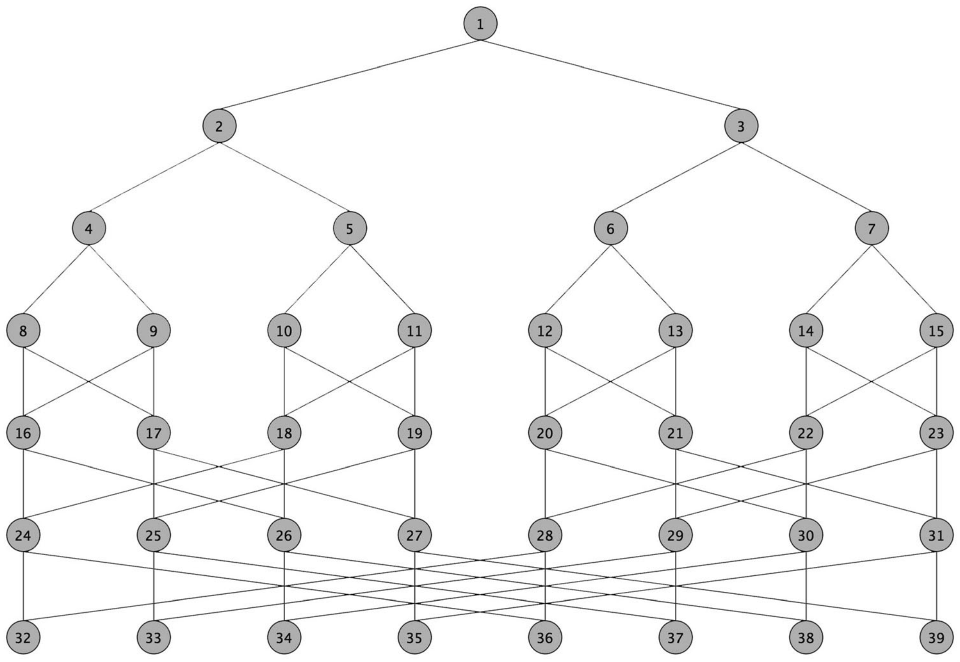


Fig. 4 Example of the fast Fourier transform parallel application with $\rho = 8$

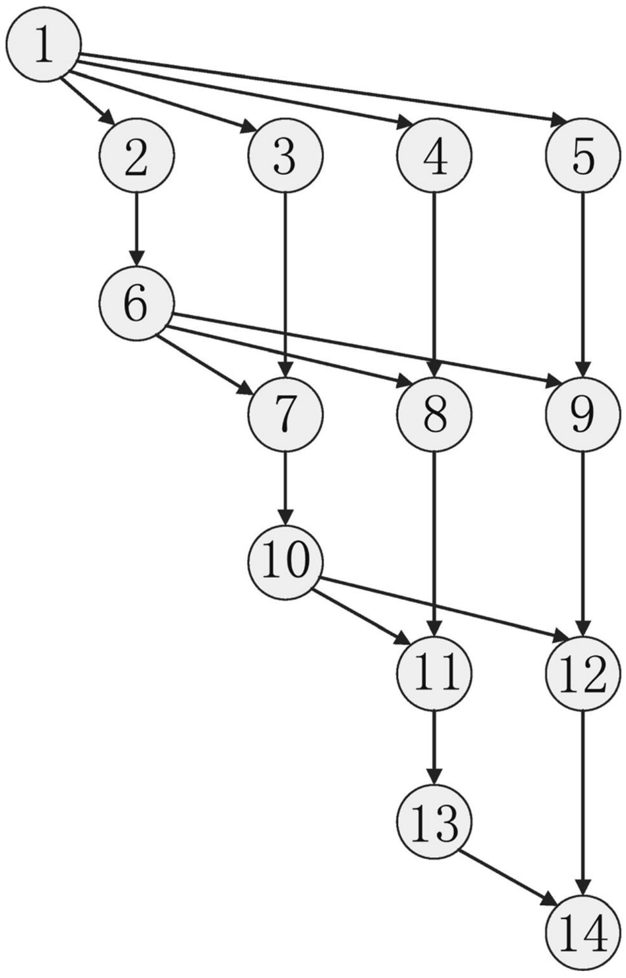


Fig. 5 Example of the Gaussian elimination parallel application with $\rho = 5$

However, the total cost obtained using DUCO is always lower than the total cost obtained using IC-PCP and DCO because of three reasons. First, HEFT minimizes the makespan instead of the minimum cost. Second, the DCO algorithm optimizes the execution cost from the upward perspective, but the IC-PCP algorithm achieves cost optimization from the downward perspective. Third, the DUCO algorithm is based on DCO to further optimize the total cost from downward and upward perspectives. The cost saving obtained with DUCO also increases with the increase in the deadline. DUCO can save more cost than DCO and IC-PCP. Specifically, when the deadline is equal to $lb(G) \times 1.4$, DUCO, in terms of cost minimization, is better than IC-PCP and DCO by 51.9% and 64.04%, respectively. These results indicate that the total execution cost of an application is decreased with the increase of the deadline. Moreover, the approach that synthetically considers “downward” and “upward” can save more execution cost than the approach that merely considers “downward” or “upward”. The superiority of the synthetic approach becomes increasingly evident when the deadline span between the deadline and lower bound is large.

Table 8 shows the actual makespan and NC of scheduling FFT parallel applications with different scales. The number of tasks is changed from 96 (small scale) to 2560 (large scale) when ρ is changed from 32 to 256. $deadline(G)$ is limited as $lb(G) \times 1.4$. The makespans obtained by the HEFT, IC-PCP, DCO, and DUCO algorithms do not exceed the user-specified deadline. The total costs using the four algorithms increase gradually with the increase in the number of tasks in applications. However, the optimized cost also increases when IC-PCP, DCO, and DUCO are used. DUCO can always save more execution cost than IC-PCP and DCO. In the best case, the optimized NC reaches 90.8%, 60.3%, and 77.11% with respect to HEFT, DCO, and IC-PCP, respectively. These results further confirm that DUCO, which is implemented from “downward” and “upward” perspectives, is more efficient than DCO and IC-PCP in cost minimization.

Table 9 shows the performance of scheduling Gaussian elimination parallel applications with varying deadline constraints. The size of the applications is limited to $\rho = 48$ (i. e., $|N| = 1175$), which is approximately equal to the number of tasks of the FFT parallel application in Table 7. $deadline(G)$ is changed from $lb(G) \times 1.0$ to $lb(G) \times 1.4$. Compared with the results in Table 7, the lower bound of

Table 7 Actual makespan and total cost of the Fast Fourier transform application with ρ=48 (i.e.,$|N|$=1152) for varying deadline constraints

$\vert N\vert$	HEFT [6]			IC-PCP [15]		DCO		DUCO	
	$cost(G)$	$lb(G)$	$deadline(G)$	$makespan(G)$	$cost(G)$	$makespan(G)$	$cost(G)$	$makespan(G)$	$cost(G)$
1152	10960	1104	1104	1104	5121	1104	10960	1104	5121
1152	10960	1104	1214	1214	4285	1204	5474	1214	2692
1152	10960	1104	1324	1324	4184	1323	3743	1324	1579
1152	10960	1104	1435	1435	3739	1413	3132	1435	1389
1152	10960	1104	1545	1545	3474	1521	2593	1545	1248

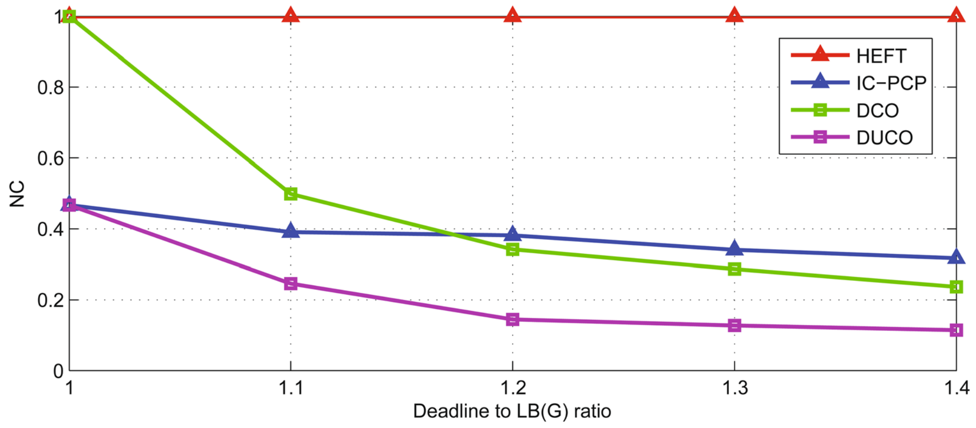


Fig. 6 Normalized cost of a fast Fourier transform application with $|N| = 1152$ at varying deadline constraints

Table 8 Normalized cost and actual makespan of fast Fourier transform applications with deadline constraints for varying numbers of tasks

|N|	HEFT [6]			IC-PCP [15]		DCO		DUCO	
	cost(*G*)	*lb*(*G*)	*deadline*(*G*)	*makespan*(*G*)	*NC*	*makespan*(*G*)	*NC*	*makespan*(*G*)	*NC*
96	723	645	903	903	0.03	901	0.031	903	0.028
224	1525	776	1086	1086	0.129	1068	0.086	1086	0.037
512	5162	940	1316	1316	0.304	1290	0.195	1316	0.096
1152	9748	1082	1514	1514	0.402	1514	0.232	1514	0.092
2560	25194	1287	1801	1801	0.401	1768	0.435	1801	0.174

Table 9 Normalized cost and actual makespan of Gaussian elimination parallel applications with $\rho = 48 (i.e., |N| = 1175)$ for varying deadline constraints

|N|	HEFT [6]			IC-PCP [15]		DCO		DUCO	
	cost(*G*)	*lb*(*G*)	*deadline*(*G*)	*makespan*(*G*)	*NC*	*makespan*(*G*)	*NC*	*makespan*(*G*)	*NC*
1175	12819	5266	5266	5266	0.243	5266	1	5266	0.243
1175	12819	5266	5792	5792	0.204	5743	0.416	5792	0.148
1175	12819	5266	6319	6319	0.2	5996	0.408	6319	0.144
1175	12819	5266	6845	6845	0.183	6703	0.35	6845	0.144
1175	12819	5266	7372	7372	0.183	7259	0.348	7372	0.132

scheduling the Gaussian elimination application is longer than the lower bound of the FFT application. However, the difference between the execution costs from scheduling the FFT application and the Gaussian elimination application is small.

Similar to the results in Table 7, our proposed DUCO can save more execution cost than IC-PCP and DCO. When $deadline(G) = lb(G) \times 1.2$, the cost saving using DUCO is 64.7% and 28.0% with regard to IC-PCP and DCO, respectively. The overall trend of Gaussian elimination and FFT applications in the same scale is similar. That is, with the increase of the deadline of the application, the total execution cost decreases gradually. These results show that DUCO is effective in different types of parallel applications.

Table 10 shows the results in scheduling Gaussian elimination parallel applications with varying numbers of tasks to further observe the performance of the proposed algorithm. The number of tasks is changed from 77 (small scale) to 1829 (large scale), which is approximately equal to the number of the FFT application in Experiment 2. $deadline(G)$ is set to $lb(G) \times 1.4$. Similar to the results in Table 8, the makespans obtained by the four algorithms do not exceed the required deadline. Moreover, the execution

Table 10 Normalized cost and actual makespan of Gaussian elimination parallel applications with deadline constraints for varying numbers of tasks

|*N*|	HEFT [6]			IC-PCP [15]		DCO		DUCO	
	cost(*G*)	*lb*(*G*)	*deadline*(*G*)	*makespan*(*G*)	*NC*	*makespan*(*G*)	*NC*	*makespan*(*G*)	*NC*
77	434	1115	1561	1561	0.046	1501	0.046	1561	0.046
299	1970	2445	3423	3423	0.107	3360	0.094	3423	0.055
665	6545	4343	6080	6080	0.109	6056	0.131	6080	0.051
1175	8624	4771	6679	6679	0.166	6678	0.211	6679	0.076
1829	20994	6582	9214	9214	0.179	8928	0.345	9214	0.08

cost increases with the increase in the number of tasks in Gaussian elimination applications. DUCO can always save more cost than IP-PCP and DCO, and the cost savings using DUCO are 76.8% and 55.3% with respect to DCO and IC-PCP in the best case ($|N| = 1829$), respectively.

After combining all results in FFT and Gaussian elimination applications, the proposed DUCO is concluded to be more efficient in cost minimization than existing algorithms, given that the required deadline constraint is satisfied in various conditions.

7 Conclusions

The cost optimization problem for deadline-constrained parallel applications in heterogeneous cloud environments is studied, and the DCO and DUCO algorithms with low-time complexity are presented. DCO is implemented by transferring the deadline constraint of the application to the deadline constraint of each task and satisfying the deadline constraint of tasks from downward perspective. DUCO is implemented from upward perspective on the basis of DCO. Furthermore, in term of cost minimization, our proposed DUCO is more efficient than existing algorithms for parallel applications in various conditions. The proposed approach can provide a theoretical basis for QoS-aware scheduling applications in heterogeneous cloud environments.

Acknowledgements This work is partially supported by the National Natural Science Foundation of China with Grant Nos. 61672217, 61602164, 61702172, the Natural Science Foundation of Hunan Province, China with Grant Nos. 2018JJ3076, 2018JJ2063, the Fundamental Research Funds for the Central Universities.

References

1. Mei, J., Li, K., Tong, Z., et al.: Profit maximization for cloud brokers in cloud computing. IEEE Trans. Parallel Distrib. Syst. **30**(1), 190–203 (2019)
2. Wang, H., Fox, K., Dongarra, G., et al.: Cloud Comptuing and Distributed System: From Parallel Processing to Web of Things. Machinery Industy Press, Beijing (2013)
3. Xie, G., Wei, Y., Le, Y., Li, R., Li, K.: Redundancy minimization and cost reduction for workflows with reliability requirements in cloud-based services. IEEE Trans. Cloud Comput. (2019). https://doi.org/10.1109/TCC.2019.2937933
4. Xie, K., Wang, X., Xie, G., et al.: Distributed multi-dimensional pricing for efficient application offloading in mobile cloud computing. IEEE Trans. Serv. Comput. **12**(6), 925–940 (2019)
5. Wu, Z., Liu, X., Ni, Z., et al.: A market-oriented hierarchical scheduling strategy in cloud workflow systems. J. Supercomput. **3**(1), 256–293 (2013)
6. Topcuoglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distri. Syst. **13**(3), 260–274 (2002)
7. Chen, Y., Xie, G., Li, R.: Reducing energy consumption with cost budget using available budget preassignment in heterogeneous cloud computing systems. IEEE Access (2018). https://doi.org/10.1109/ACCESS.2018.2825648
8. Zhou, A., He, B.: Transformation-based monetary cost optimizations for workflows in the cloud. IEEE Trans. Cloud Comput. **2**(1), 85–98 (2014)
9. Arabnejad, V., Bubendorfer, K., Ng, B.: Scheduling deadline constrained scientific workflows on dynamically provisioned cloud resources. Future Gener. Comput. Syst. **75**, 348–364 (2017)
10. Deldari, A., Naghibzadeh, M., Abrishami, S.: CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud. J. Supercomput. **73**(2), 1–26 (2016)
11. Liu, J., Li, K., Yang, Q., et al.: Minimizing cost of scheduling tasks on heterogeneous multi-core embedded systems. ACM Trans. Embed. Comput. Syst. **16**(2), 36 (2016)
12. Abrishami, S., Naghibzadeh, M., Epema, D.: Cost-driven scheduling of grid workflows using partial critical paths. IEEE Trans. Parallel Distrib. Syst. **23**(8), 1400–1414 (2012)
13. Singh, S., Chana, I.: A survey on resource scheduling in cloud computing: issues and challenges. J. Grid Comput. **14**(2), 217–264 (2016)
14. Xie, G., Li, Y., Xie, Y., et al.: Recent advances and future trends for automotive functional safety design methodologies. IEEE Trans. Ind. Inform. **16**(96), 5629–5642 (2020)
15. Abrishami, S., Naghibzadeh, M., Epema, D.: Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds. Future Gener. Comput. Syst. **29**(1), 158–169 (2013)
16. Mao, M., Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Proceedings of 2011 international conference for high performance computing, networking, storage and analysis. ACM, p. 49 (2011)

17. Rodriguez, M.A., Buyya, R.: Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans. Cloud Comput. **2**(2), 222–235 (2014)
18. Malawski, M., Juve, G., Deelman, E., et al.: Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. Future Gener. Comput. Syst. **48**, 1–18 (2015)
19. Tian, G., Xiao, C., Xie, J.: Scheduling and fair cost-optimizing methods for concent multiple DAGs with deadline sharing resources. Chin. J. Comput. **37**(7), 1067–1619 (2014)
20. Mortazavi-Dehkordi, M., Zamanifar, K.: Efficient deadline-aware scheduling for the analysis of Big Data streams in public Cloud. Clust. Comput. **23**(1), 241–263 (2020)
21. Ju, Y., Buyya, R., Tham, C. K.: QoS-based scheduling of workflow applications on service grids. In: Proceedings of 1st IEEE international conference on e-science and grid computing (2005)
22. Tian, G., Xiao, C., Xu, Z., et al.: Hybrid scheduling strategy for multiple DAGs workflow in heterogeneous system. J. Softw. **23**(10), 2720–2734 (2012)
23. Bittencourt, L., Madeira, E.: HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds. J. Internet Serv. Appl. **2**(3), 207–227 (2011)
24. Ahmad, W., Alam, B., Ahuja, S., et al.: A dynamic VM provisioning and de-provisioning based cost-efficient deadline-aware scheduling algorithm for Big Data workflow applications in a cloud environment. Clust. Comput. (2020). https://doi.org/10.1007/s10586-020-03100-7
25. Wu, C.Q., Lin, X., Yu, D., et al.: End-to-end delay minimization for scientific workflows in clouds under budget constraint. IEEE Trans. Cloud Comput. **3**(2), 169–181 (2015)
26. Arabnejad, H., Barbosa, J., Prodan, R.: Low-time complexity budget-deadline constrained workflow scheduling on heterogeneous resources. Future Gener. Comput. Syst. **55**, 29–40 (2016)
27. Arabnejad, H., Barbosa, J.: Multi-QoS constrained and Profit-aware scheduling approach for concurrent workflows on heterogeneous systems. Future Gener. Comput. Syst. **78**, 402–412 (2018)
28. Chen, W., Xie, G., Li, R., et al.: Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems. Comput. Syst, Future Gener (2017). https://doi.org/10.1016/j.future.2017.03.008
29. Rodriguez, M., Buyya, R.: Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans. Cloud Comput. **2**(2), 222–235 (2014)
30. Convolbo, M., Chou, J.: Cost-aware DAG scheduling algorithms for minimizing execution cost on cloud resources. J. Supercomput. **72**(3), 1–28 (2016)
31. Liu, Z., Wang, S., Sun, Q., et al.: Cost-aware cloud service request scheduling for SaaS providers. J. Beijing Univ. Posts Telecommun. **57**(2), 291–301 (2013)
32. Alkhanak, E., Lee, S., Rezaei, R., et al.: Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. J. Syst. Softw. **113**, 1–26 (2016)

Weihong Chen is a Ph.D. candidate at the College of Computer Science and Electronic Engineering, Hunan University, and a professor at Hunan City University. She received her Master's degree from Hunan University in 2006. Her research interest covers cyber physical systems, distributed computing, and machine learning.

Guoqi Xie received his Ph.D. degree in computer science and engineering from Hunan University, China, in 2014. He was a Postdoctoral Researcher at Nagoya University, Japan, from 2014 to 2015. Since 2015 He is working as a Postdoctoral Researcher at Hunan University, China. He has received the best paper award from ISPA 2016. His major interests include embedded and real-time systems, parallel and distributed systems, software engineering and methodology. He is a member of IEEE, ACM, and CCF.

Renfa Li is a Professor of computer science and electronic engineering, and the Dean of College of Computer Science and Electronic Engineering, Hunan University, China. He is the Director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. His major interests include computer architectures, embedded computing systems, cyber-physical systems, and Internet of things. He is a member of the council of CCF, a senior member of IEEE, and a senior member of ACM.

Keqin Li is a SUNY Distinguished Professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published over 550 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, IEEE Transactions on Services Computing, IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.