Information Sciences 179 (2009) 2562–2575

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Deadline division-based heuristic for cost optimization in workflow scheduling

Yingchun Yuan [a,b,*], Xiaoping Li [a,c], Qian Wang [a,c], Xia Zhu [a,c]

[a] School of Computer Science and Engineering, Southeast University, Nanjing 210096, PR China
[b] Faculty of Information Science and Technology, Agriculture University of Hebei, Baoding 071001, PR China
[c] Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing 210096, PR China

ARTICLE INFO

Article history:
Received 1 August 2007
Received in revised form 20 May 2008

Keywords:
Grid computing
Workflow
Directed acyclic graph
Cost/time tradeoff
Heuristic

ABSTRACT

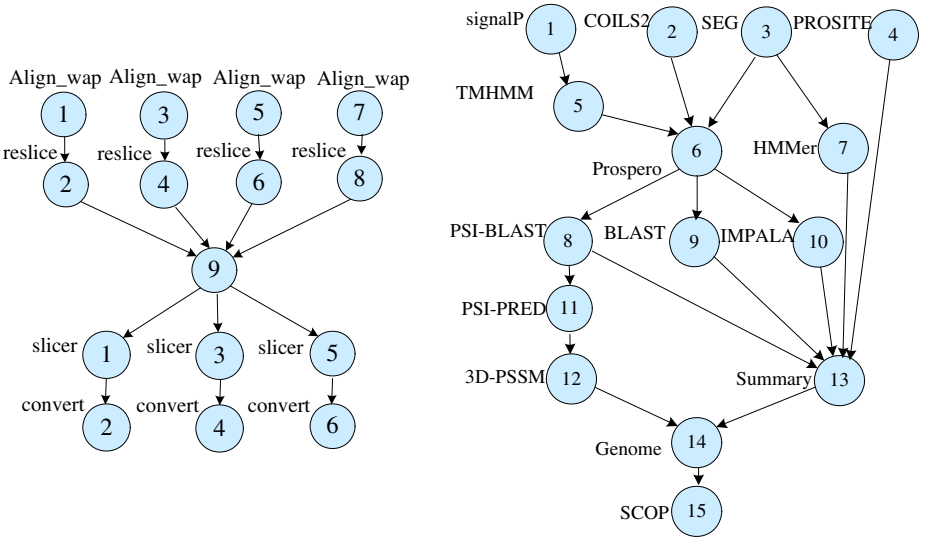
Cost optimization for workflow applications described by Directed Acyclic Graph (DAG) with deadline constraints is a fundamental and intractable problem on Grids. In this paper, an effective and efficient heuristic called DET (Deadline Early Tree) is proposed. An early feasible schedule for a workflow application is defined as an Early Tree. According to the Early Tree, all tasks are grouped and the Critical Path is given. For critical activities, the optimal cost solution under the deadline constraint can be obtained by a dynamic programming strategy, and the whole deadline is segmented into time windows according to the slack time float. For non-critical activities, an iterative procedure is proposed to maximize time windows while maintaining the precedence constraints among activities. In terms of the time window allocations, a local optimization method is developed to minimize execution costs. The two local cost optimization methods can lead to a global near-optimal solution. Experimental results show that DET outperforms two other recent leveling algorithms. Moreover, the deadline division strategy adopted by DET can be applied to all feasible deadlines.

1. Introduction

Workflow applications are generally described as collections of tasks to be processed in a well-defined order to accomplish a specific goal [26]. Many complex applications in e-science and e-business can be modeled as workflows [12,29]. Because of the large amounts of computation and data involved, these workflows require the power of the Grid to run efficiently. Grid computing [13] has emerged as the next generation computing platform for solving large-scale problems in science, engineering, and commerce. It facilitates the sharing and aggregation of heterogeneous and distributed resources, such as computing resources, data sources, instruments, and application services. Grid workflow can be defined as the composition of different Grid application services that run on heterogeneous and distributed resources in a well-defined order to accomplish a specific goal [28]. Yu and Buyya [29] proposed a taxonomy that characterizes and classifies various approaches for building and executing Grid workflows. A popular representation of workflow applications is Directed Acyclic Graph (DAG), in which nodes represent individual tasks and directed arcs or edges represent inter-task data or control dependencies. DAG-based workflow applications usually have different structures, such as pipeline, parallel and hybrid structures. A pipeline application executes a number of tasks in a single sequential order. A parallel application requires multiple pipelines to be executed in parallel. Fig. 1a shows a neuroscience workflow [33] where there are four pipelines (1–2, 3–4, 5–6 and 7–8) before task 9. A hybrid structure application is a combination of both parallel and sequential applications. For example, Fig. 1b shows a protein annotation workflow [25] developed by the London e-Science Centre.

* Corresponding author. Address: School of Computer Science and Engineering, Southeast University, Nanjing 210096, PR China. Tel.: +86 2583790901.
E-mail addresses: nd_hd_yyc@163.com (Y. Yuan), xpli@seu.edu.cn (X. Li), qwang@seu.edu.cn (Q. Wang).

0020-0255/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2009.01.035



(a) Parallel application (b) Hybrid structure

Fig. 1. Workflow applications with different structures (the label on the left of a node denotes the required service type).

Workflow scheduling is one of the key problems on Grids. It is meant to deal with the allocation of tasks to suitable resources so that the objective function can be minimized or maximized while maintaining task-precedence constraints. A number of Grid workflow management systems with scheduling algorithms have been developed by several projects, such as Condor DAGMan [16], GrADS [8] and Pegasus [5,12]. They facilitate the execution of workflow applications and minimize their execution time on Grids. However, Grid resources are normally heterogeneous in terms of their architectures, powers, configurations, and availabilities. They are owned and managed by different organizations with different access policies and cost models that vary with time and the users. The resource owners and end-users have different goals, objectives, and demand patterns. Due to these unique characteristics, different and effective performance parameters (i.e., quality of service), such as time, cost, and reliability [24], are applied to reflect the actual characteristics of Grids. QoS (quality of service) is a determinant factor used to ensure customer satisfaction that can be expressed as the utility functions over QoS attributes or the QoS constraints. Therefore, QoS-based workflow scheduling becomes a significant and challenging problem. This paper aims to study the QoS optimization strategy for workflow scheduling on Grids.

The remainder of the paper is structured as follows. Section 2 introduces the related work. Section 3 formulates the workflow scheduling problem. An early feasible schedule and its related definitions are introduced in Section 4. Section 5 reviews the leveling algorithms and describes the proposed algorithm, DET. In Section 6, experimental results are given, followed by conclusions in Section 7.

2. Related work

Currently, QoS-based scheduling is an active research area on Grids. Foster et al. [15] described a general-purpose architecture for reservation and allocation (GARA) that supports flow-specific QoS specification. Al-Ali et al. [3,4] extended QoS properties to the service abstraction of OGSA (Open Grid Service Architecture)[14]. The realization of QoS requires mechanisms such as advance or on-demand reservation of resources. Ǒogan and Ozguner [9] considered the scheduling problem of a set of independent tasks with multiple QoS requirements, including timeliness, reliability, and security. Golconda and Ozguner [17] compared five QoS-based scheduling heuristics for independent tasks in terms of three parameters: the number of satisfied users, makespan, and total utility of the meta-task. Buyya et al. [7] proposed a distributed computational economy-based framework called GRACE (Grid Architecture for Computational Economy). They also developed three heuristic scheduling algorithms for cost, time, and time-variant optimization problems with deadline and budget constraints [1,6]. Li and Li [21,22] presented QoS optimization strategies for multi-criteria scheduling problems. As opposed to the independent-tasks scheduling problem, this paper aims to schedule workflow applications described by a Directed Acyclic Graph.

QoS-based Service selection is also popular. Considering multi-dimensional QoS properties such as price, duration, reliability, availability, and reputation, Zeng et al. [32] formulated the properties as weighted objective functions. For the service composition model described by the statechart, they also presented an integer programming method that addresses the issue of dynamic service selection based on user satisfaction. Gu and Nahrstedt [18] and Xu and Nahrstedt [27] proposed global planning algorithms for dynamic QoS-aware service composition, but the underlying service composition model does not support parallelism or branching. Jin et al. [19] considered service composition represented by DAG on Grids. A multi-dimensional QoS model was presented, and heuristics based on simulated annealing were established to minimize the

performance/price of service composition under deadline and budget constraints. As opposed to the previously mentioned algorithms, this paper considers two primary factors (time and cost) that users are generally concerned with, and focuses on the cost optimization for Grid workflow scheduling with deadline constraints.

Many time-cost optimization methods have been proposed for scheduling DAG-based Grid workflow applications. Yu et al. [30] proposed a deadline division strategy called Deadline Top Level (DTL) for scheduling scientific workflow applications with deadline constraints. Workflow tasks are first grouped based on their depths in the graph, and then the whole deadline is divided into level deadlines. All tasks in the same level have the same start time and sub-deadline. DTL is a very simple but efficient heuristic. Lin and Lin [23] conceptualized the Grid network into a prioritized CPM network when the number of gird services is large enough to accommodate the number of tasks that need to be run simultaneously at any given time. They identified the time-cost optimization problem for DAG-based Grid applications as the discrete time-cost tradeoff problem (DTCTP) in project scheduling [11]. DTCTP has been investigated by numerous researchers in project management. De et al. [11] presented an excellent review for DTCTP, and proved that the problem is a strongly NP-hard problem for general project networks [10]. Currently, the best-known algorithms for solving the optimization problem rely on dynamic programming and the branch and bound method, but they generally suffer from computational limitations. Heuristic procedures for the DTCTP have been sparse. Akkan et al. [2] provided lower and upper bounds using column generation techniques based on network decomposition. Efficient and effective heuristics are very suitable for dynamic Grid environments. According to Yu and Lin, we have investigated a heuristic called Deadline Bottom Level (DBL) [31]. By analyzing the parallel and synchronization properties of DAG, all tasks are partitioned into *BL* (Bottom Level) groups using a backward method. The workflow deadline is segmented into the time intervals of all tasks. All tasks in each bottom level have the same sub-deadline, but the starting time of a task in each level is determined by the maximum finish time of its predecessors, rather than the finish time of its parent group which is adopted by DTL. The DBL algorithm can considerably improve the average performance of DTL. Although the leveling algorithms (DBL and DTL) are very simple and relatively effective, the temporal relationships may be partially changed, and workflow applications with shorter deadline constraints cannot be optimized effectively. In addition, two leveling algorithms do not consider the sequential characteristics of DAG.

In this paper, a heuristic called Deadline Early Tree (DET) is proposed. The Early Tree (ET) is defined. According to the Early Tree, the Critical Path is given and workflow tasks are grouped. For critical activities, the optimal cost solution can be obtained by a dynamic programming method. The whole deadline is also segmented into time windows in terms of the slack time float. For non-critical activities, an iterative procedure is proposed to maximize time windows while keeping the precedence constraints among activities. A local cost optimization method is applied to non-critical activities within the allocated time windows. The performance and efficiency of the proposed approach are evaluated and compared to two leveling algorithms.

3. Problem description

Many complex applications in e-science and e-business can be modeled as workflows, which can be described by Directed Acyclic Graph (DAG). Let $G = \{V, E\}$ denote the DAG, where the node set $V = \{1, 2, \ldots, n\}$ denotes the set of tasks (or activities), and the directed arc (edge) set E represents inter-task data or control dependencies. Assume these nodes are topologically numbered such that an arc always leads from a smaller to a higher node number, i.e., $i < j$ if task i is a predecessor of task j. Task 1 and n are dummy single-start and single-terminal nodes (the processing time and execution cost of dummy node are equal to 0). Each arc $(i,j) \in E$ represents the precedence constraint from i to j, i.e., task j cannot start before i has finished. The DAG representation of the workflow example in Fig. 1b is shown in Fig. 2, where node 1 and node n are two dummy nodes.

Consider a Grid network with many computing services, including CPU, bandwidth, data access, software and other hardware. Each computing service can provide many service levels with differentiated service qualities, i.e., multiple services provide similar functionality but with different non-functional properties, such as processing time, cost, and reliability. Quality of Service (QoS) becomes a key factor to differentiate these services. Thus, for any activity i in graph G, there are many candidate services that can fulfill it. All candidate services for i are called the service pool (SP) of activity i, denoted as $SP(i)$. Let $l(i)$ denote the service pool length, i.e., $l(i) = |SP(i)|$. Service pools of dummy activities are empty. In the same service pool, different services have different processing times and costs. Generally, the faster a service is, the more a user pays [30]. Let S_{ij} be the jth service name for conducting activity i, t_{ij} be the processing time on service S_{ij}, and c_{ij} be the cost for executing

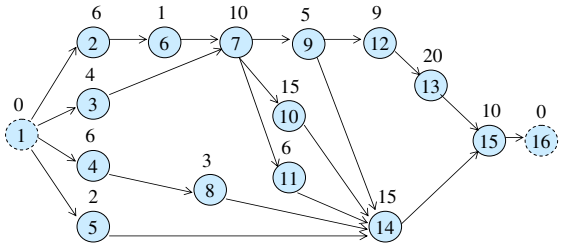


Fig. 2. DAG representation of the workflow application sample in Fig. 1b.

Table 1
The service pool example for Fig. 2.

Service name	Cost	Time	Service name	Cost	Time
$S_{2,1}$	6	10	$S_{9,2}$	18	5
$S_{2,2}$	8	8	$S_{10,1}$	100	30
$S_{2,3}$	11	6	$S_{10,2}$	150	20
$S_{3,1}$	10	5	$S_{10,3}$	200	15
$S_{3,2}$	12	4	$S_{11,1}$	50	10
$S_{4,1}$	5	6	$S_{11,2}$	80	6
$S_{5,1}$	10	4	$S_{12,1}$	18	9
$S_{5,2}$	15	2	$S_{13,1}$	40	25
$S_{6,1}$	5	3	$S_{13,2}$	50	20
$S_{6,2}$	10	2	$S_{14,1}$	80	30
$S_{6,3}$	20	1	$S_{14,2}$	120	20
$S_{7,1}$	25	15	$S_{14,3}$	150	15
$S_{7,2}$	30	10	$S_{15,1}$	50	13
$S_{8,1}$	30	3	$S_{15,2}$	60	10
$S_{9,1}$	14	8			

i on service s_{ij}. Thus, the 3-tuple (S_{ij}, t_{ij}, c_{ij}) denotes the jth service that can fulfill activity i. The service pool of activity i can be expressed as $SP(i) = \{(S_{ij}, t_{ij}, c_{ij}) | 1 \leqslant j \leqslant l(i)\}$. If one lets $SP(G)$ denote the service pool of graph G, then $SP(G) = \{SP(i) | i \in V\}$. Table 1 shows a service pool example for Fig. 2.

Let δ_n be the given time constraint (deadline), which is also the latest completion time of task n. The objective of this problem is to select an appropriate service for each task in a workflow application so that the total cost can be minimized while meeting the given deadline.

Assume that the number of gird services is large enough to accommodate the number of tasks that need to be run simultaneously at any given time, so that the waiting time for an available service is negligible. Thus, each service is able to provide the expected response time and execution cost. For each kind of service allocation to all workflow tasks, there exists a corresponding critical path. The workflow completion time is the duration of the critical path, and the total cost is the sum of the prices of all of the tasks.

Since the service selection is associated with the service cost and total completion time, the above problem is transformed into the time-cost tradeoff problem [11]. Because each candidate service with a shorter responding time corresponds to a higher price, the service selection is identical to the discrete time–cost tradeoff problem. Let f_i $(1 \leqslant i \leqslant n)$ be the completion time of task i, and let x_{ik} be a 0–1 variable that is 1 if service k is selected for executing activity i and 0 otherwise. The optimization problem can be described by the following formulation

$$\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{1 \leqslant k \leqslant l(i)} c_{ik} x_{ik} \\
\text{s. t.} \quad & \sum_{k=1}^{l(i)} x_{ik} = 1, \quad i \in V \\
& f_i \leqslant f_j - \sum_{1 \leqslant k \leqslant l(i)} t_{ik} x_{ik}, \quad \forall (i,j) \in E \\
& f_n \leqslant \delta_n \\
& x_{ik} \in \{0,1\}, \quad i \in V,\ 1 \leqslant k \leqslant l(i)
\end{aligned}$$

The objective function $\min \sum_{i \in V} \sum_{1 \leqslant k \leqslant l(i)} c_{ik} x_{ik}$ corresponds to minimizing the workflow total cost. $\sum_{k=1}^{l(i)} x_{ik} = 1, i \in V$ ensures that exactly one service should be chosen for each activity. The constraints $f_i \leqslant f_j - \sum_{1 \leqslant k \leqslant l(i)} t_{ik} x_{ik}, \forall (i,j) \in E$ maintain the precedence constraints among the activities. Constraint $f_n \leqslant \delta_n$ guarantees that the workflow will be completed by its deadline. Constraint $x_{ik} \in \{0,1\}$, $i \in V$, $1 \leqslant k \leqslant l(i)$ guarantees that x_{ik} is a 0–1 variable.

The traditional time-cost tradeoff problem was proved to be strongly NP-hard in general project networks [10]. In addition, Grid is a highly dynamic, distributed environment. Therefore, it is essential to investigate effective and efficient algorithms for workflow instances, which are common in practice.

4. An early feasible schedule

For $\forall i \in V$, let the 3-tuple (S_i^*, t_i^*, c_i^*) denote the fastest service in service pool $SP(i)$; then t_i^* should hold such that

$$t_i^* = \min_{1 \leqslant j \leqslant l(i)} \{t_{ij}\}, \quad \text{for } \forall (S_{ij}, t_{ij}, c_{ij}) \in SP(i)$$

In Fig. 2, the number above each node denotes the minimum processing time of the activity. The earliest start time and finish time of each activity can be computed based on traditional forward pass calculations. Let s_i be the earliest start time and f_i be the earliest finish time of activity i. The statement $s_j - s_i \geqslant t_i^*$ is true for $\forall (i,j) \in E$. An early feasible schedule can be obtained if all of the activities start with the earliest time.

Definition 1. A spanning tree T is the sub-graph of G with $n-1$ arcs and no loops. A tree is called an Early Tree (ET) if it is associated with an early feasible schedule, which can be formulated as follows

$$\begin{cases} s_1 = 0 \\ s_j - s_i = t_i^*, \quad \forall (i,j) \in E_T \\ s_j - s_i \geqslant t_i^*, \quad \forall (i,j) \in E \setminus E_T \end{cases}$$

where E_T denotes the edges of the Early Tree, and $E \backslash E_T$ is the edge subset of E obtained by deleting E_T.

If one lets $pred(i)$ denote the immediate predecessors of activity i, then the construction procedure of the Early Tree can be written as follows.

Early Tree Generation Algorithm (ETGA)

1. $E_T \leftarrow NULL$, $s_1 \leftarrow 0$, $f_1 \leftarrow 0$
2. for $i = 2$ to n
   - 2.1. $f_k \leftarrow \max\{f_j | j \in pred(i)\}$;
   - 2.2. $s_i \leftarrow f_k$;
   - 2.3. $f_i \leftarrow s_i + t_i^*$;
   - 2.4. $E_T \leftarrow E_T \cup (k,i)$.
3. Output E_T, s_i, $f_i (1 \leqslant i \leqslant n)$;

The Early Tree contains all activities. For the graph in Fig. 2, the Early Tree constructed by ETGA is shown in Fig. 3, in which entry node 1 is the root of the ET. All nodes with no successors are called leaves, such as 3, 5, 8, 11, 14 and 16. For any activity $i (1 \leqslant i \leqslant n)$, its time interval can be determined by its earliest start time s_i and its earliest finish time f_i, i.e., $[s_i, f_i]$.

Definition 2. A path of the Early Tree is composed of all activities from the root node to some leaf. The path length is the sum of the processing time of all activities on the path.

The number of paths is actually the number of leaves of the ET. Let L_{ET} be the set of leaves and $p(j)$ be the node set including all activities from the root to leaf j. The path set of the ET, denoted as P_{ET}, can be given by

$$P_{ET} = \{p(j) | j \in L_{ET}\}$$

If one lets $C_{p(j)}$ denote the path length of $p(j)$, then its path length is also equal to the earliest completion time of leaf j. For example, the path set of the ET in Fig. 3 is $P_{ET} = \{p(8) = \{1,4,8\}, p(11) = \{1,2,6,7,11\}, p(14) = \{1,2,6,7,10,14\}, p(3) = \{1,3\}, p(5) = \{1,5\}, p(16) = \{1,2,6,7,9,12,13,15,16\}\}$, and the path lengths of all these paths are 9, 23, 47, 4, 2 and 61, respectively.

Definition 3. The path with the longest path length in P_{ET} is called the Critical Path. All activities on the Critical Path are called critical activities.

Since exit node n of the Early Tree has the maximum finish time, $p(n)$ is the Critical Path of the ET. For the example in Fig. 3, the Critical Path is $p(16)$, and both dummy activity 1 and dummy activity 16 are on the Critical Path.

The Early Tree corresponds to an early feasible schedule solution with the minimum completion time.

Definition 4. The difference between the deadline and the Critical Path length is the workflow total float.

The workflow total float (denoted as W_{TF}) is the maximum delay for workflow execution. It can be calculated by

$$W_{TF} = \delta_n - C_{p(n)}$$

Obviously, the workflow total float should satisfy $W_{TF} \geqslant 0$, i.e., a feasible deadline δ_n should be greater than or equal to the workflow earliest completion time $C_{p(n)}$.

In terms of the above statements, the workflow cost may be decreased by forcing its completion time to the given deadline.

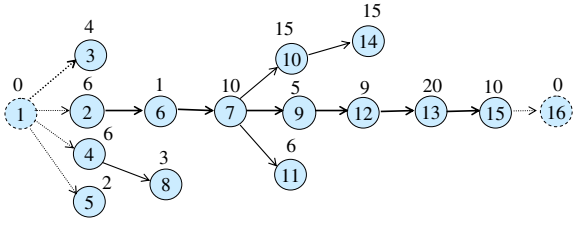


Fig. 3. An early tree for the graph in Fig. 2.

5. Proposed heuristic based on the Early Tree

As described previously, the cost optimization problem for workflow scheduling with a deadline constraint is generally NP-hard. A sub-optimal solution can be obtained by local optimization strategies. Two leveling algorithms (DBL and DTL) are able to partition the tasks into different levels in terms of the parallel and synchronization properties. All tasks in the same level have the same sub-deadline. As opposed to leveling-based deadline division strategies, the DET algorithm partitions all tasks into different paths based on the Early Tree. The whole deadline is divided into time windows of critical activities, which can be applied to all feasible deadlines. An iterative algorithm is proposed to determine the time windows of non-critical activities while keeping the precedence constraints among tasks. For critical activities and a set of sequential tasks, the execution costs are locally optimized based on a dynamic programming strategy, which can improve the performance. Before introducing the details of the DET algorithm, we first review the two leveling algorithms.

5.1. Leveling algorithms

DTL [30] and DBL [31] are two heuristics that optimize the workflow total cost using the deadline division strategy, and they consist of three same phases: task leveling, deadline assignment, and service selection. The difference between DTL and DBL is that all tasks are grouped by different leveling methods. In the following, the main steps of DBL are shown.

5.1.1. Task leveling

The Bottom Depth (BD) of a node is defined as the maximum number of edges along any path from the node to the exit node in graph G. Let $BD(i)$ denote the bottom depth of task i; it is computed recursively by traversing the graph upward starting from the exit node. The exit node has a bottom depth of zero. Then

$$BD(i) = \begin{cases} 0, & i = n \\ \max\limits_{j \in succ(i)} \{BD(j)\} + 1, & \text{otherwise} \end{cases}$$

where $succ(i)$ is the set of immediate successors of task i. All tasks with the same bottom depth can be grouped into a bottom level. Let BL_k be the kth bottom level. It can be calculated by

$$BL_k = \{i | BD(i) = k, i \in V\}$$

All levels are sorted by their bottom depths. Then, the list length (denoted as G_L) is $BD(1) + 1$, where the first line is $BL_{BD(1)}$ and the last line is $BL_{BD(n)}$. For any BL_k ($BD(n) < k < BD(1)$), its predecessor level is BL_{k+1} and its successor level is BL_{k-1}.

5.1.2. Deadline assignment

Those tasks in the same bottom level have parallel and synchronization characteristics. Therefore, all tasks in each level are assigned the same sub-deadline so that they still preserve the synchronization finish characteristics. On the other hand, the start execution time of each task in the same level can be computed according to its parent tasks. Thus, the time intervals of each task can be determined. When each task selects the fastest service for execution, the Level deadline δ_{BL_i}, the start time β_j, and the finish time δ_j of task j can be computed by

$$\begin{cases} \beta_j = 0, j = 1 \\ \delta_{BL_i} = \max\left\{\beta_j + \min\limits_{1 \leqslant k \leqslant l(j)} \{t_{jk}\}\right\}, \quad j \in BL_i \\ \delta_j = \delta_{BL_i}, \quad j \in BL_i \\ \beta_j = \max\{\delta_i\}, \quad i \in pred(j) \end{cases} \tag{1}$$

Eq. (1) may change the Critical Path because it may delay the partial tasks' start times and finish times. Therefore, the earliest completion time of the workflow may also be delayed. Thus, the earliest completion time determined by Eq. (1) is called the Bottom Level completion time, which is denoted as $BL_{\min}$. It is also equal to the level deadline of level BL_0, i.e., $BL_{\min} = \delta_{BL_0}$.

For a given deadline δ_n, if $\delta_n \geqslant BL_{\min}$, then the whole deadline can be divided into level deadlines by Eq. (1). The division strategy not only meets the deadline constraint, but the time interval for each task can also be further enlarged by allocating the leveling time float to each level equally. Let a_{LS} denote the level float. Then $a_{LS} = (\delta_n - BL_{\min})/(G_L - 2)$(except for two dummy levels consisting of dummy activities). Thus, Eq. (1) can be transformed into the following equation:

$$\begin{cases} \beta_j = 0, j = 1 \\ \delta_{BL_i} = \max\limits_{j \in BL_i} \left\{\beta_j + \min\limits_{1 \leqslant k \leqslant l(j)} \{t_{jk}\}\right\} + a_{LS}, \quad i \neq 0, i \neq BD(1) \\ \delta_j = \delta_{BL_i}, \quad j \in BL_i \\ \beta_j = \max\{\delta_i\}, \quad i \in pred(j) \end{cases} \tag{2}$$

According to Eq. (2), each task can select the cheaper service resource within its time windows. However, the strategy is not suitable for shorter deadlines, i.e., $C_{p(n)} \leqslant \delta_n < BL_{\min}$, which can be solved by the Minimum Critical Path Method (MCP)[31].

5.1.3. Service selection

After deadline distribution, each activity can be mapped to the most appropriate service by

$$\min_{1\leqslant k\leqslant l(i)} \{c_{ik} | s_i \leqslant t_{ik} \leqslant f_i, (S_{ik}, t_{ik}, c_{ik}) \in SP(i)\}, \quad \text{for } \forall i \in V \tag{3}$$

If each service selection guarantees that the task execution can be completed within its time windows, the whole workflow execution can be completed within the whole deadline. Similarly, the cost minimization solution for each task leads to an optimized cost solution for the entire workflow. Therefore, an optimized workflow schedule can be constructed by all local optimal schedules.

The DTL algorithm has almost the same strategy as DBL, in which all tasks are grouped by their Top Levels. The Top Level (TL) of a node is defined as the maximum number of edges along any path from the entry node to the node in graph G. In addition, it mandates that all tasks in same top-level have the same start time and finish time.

5.2. Deadline Early Tree (DET) algorithm

For the deadline division strategy, it is crucial to divide the whole deadline into task deadlines while maintaining the precedence constraints among activities and meeting the given deadline. At the same time, the time window for each activity should be enlarged such that each task can select a cheaper service to decrease the total cost. In addition, local cost optimization strategies should be applied to critical activities as well as those sequential non-critical activities. Thus, the performance of the deadline division strategy can be effectively improved. This section introduces a new deadline division strategy based on Early Tree. It consists of two phases: cost optimization for critical activities, and cost optimization for non-critical activities.

5.2.1. Cost optimization for critical activities

In the Early Tree, the time interval $[s_j, f_j]$ for task j implies an initial time window in which the task j has to be processed. As described previously, the given deadline is generally greater than the workflow earliest completion time, $C_{p(n)}$. By facilitating the total float, the total cost can be decreased. The time window of each activity can be extended so that one can select a cheaper service to decrease the workflow total cost.

It is known that the given deadline constraint is also the latest completion time of the Critical Path. Therefore, the time-cost optimization problem for the Critical Path is a special optimization problem for purely serial cases, which can be transformed into a multi-phase decision-problem based on dynamic programming.

Let the critical activity set denote CP, and p be the number of critical activities, then $CP = \{1, 2, \ldots, p\}$. Let $f(i,s)$, $i \in CP$ be the minimum cost of performing task i such that task i starts at time s. Without generality, $s \in [ES_i, LS_i]$ is restricted to be an integer. The earliest starting time ES_i for task i can be computed based on traditional forward critical path calculations. Subsequently, for the given deadline δ_n, its latest starting time LS_i is computed by a backward pass (at this time, each task has been allocated the fastest service). Then, the service selection method for critical activities is listed below.

Service selection for critical activities (SSCA)

1. In the first phase, the minimum cost of the last activity is computed by

$$f(p,s) = \min_{1\leqslant j\leqslant l(p)} \{c_{pj}\}, \quad ES_p \leqslant s \leqslant LS_p,\ s + t_{pj} < \delta_n$$

2. At phase k, the minimum cost associated with task i that starts at time s can be computed by

$$f(i,s) = \min_{1\leqslant j\leqslant l(i)} \{f(i+1, s+t_{ij}) + c_{ij}\}, \quad ES_i \leqslant s \leqslant LS_i,\ i < p$$

In the last phase, the minimum cost at any valid time for critical activities can be computed. Those service selections with the minimum cost correspond to the optimal cost solution.

The dynamic programming method indicates the best services for executing critical activities. Its earliest completion time is also the workflow earliest completion time. Then the workflow earliest completion time, denoted as $C_{\max}$, is the sum of the processing times of all selected services. It must be less than or equal to the deadline, i.e., $C_{\max} \leqslant \delta_n$.

After determining the service selections for critical activities, the whole deadline can be transformed into the time windows for critical activities. If $C_{\max} < \delta_n$, the slack time float $(\delta_n - C_{\max})$ can be equally allocated to critical activities. Let a_s denote the critical activity float, which is defined as the maximum delay in the activity finish time. Then $a_s = (\delta_n - C_{\max})/(|p(n)| - 2)$(except for two dummy activities). Let $S_{ij'}$ be the selected service for activity i. The time windows of critical activities can be computed by

$$\begin{cases} s_1 = f_1 = 0 \\ f_i = s_i + t_{ij'} + a_s \\ s_j = f_i \\ s_n = f_n = \delta_n \end{cases}, \quad \forall i,j \in p(n),\ (i,j) \in E_T \tag{4}$$

At the same time, the time windows for non-critical activities should be adjusted to preserve the feasible schedule of the Early Tree. Therefore, a post-processing method is investigated to accomplish the above operations. Let *pred*(i) and *succ*(i) be the set of immediate predecessors and the set of immediate successors of activity i, respectively. The post-processing method is described as follows.

Post processing for critical activities (PPCA)

1. Determine the time windows for critical activities via Eq. (4);
2. Build the new spanning tree by delaying the start time and finish time of non-critical tasks, denoted as *CT* (Current Tree). Let E_{CT} be the arc set of Current Tree *CT*, then
   2.1. $E_T \leftarrow NULL$, $s_1 \leftarrow 0$, $f_1 \leftarrow 0$;
   2.2. for $i = 2$ to $n-1$
      If $i \in p(n)$, then
        Find its predecessor in $p(n)$, denoted as k, $E_{CT} \leftarrow E_{CT} \cup (k,i)$;
      Else
        $f_k \leftarrow \max\{f_j | j \in pred(i)\}$, $s_i \leftarrow f_k$, $f_i \leftarrow s_i + t_i^*$, $E_{CT} \leftarrow E_{CT} \cup (k,i)$;
3. Computing path set of Current Tree *CT*, denoted as P_{CT};
   3.1. Find all leaves in *CT*, denoted as L_{CT}
   3.2. For $(\forall j \in L_{CT})$
      Search the path from leaf j to the root 1 in reverse order;
      All nodes from the root node to leaf j consist of path $p(j)$;
      Add path $p(j)$ to set P_{CT};
4. For $(\forall p(j) \in P_{CT})$/* Update set P_{CT} such that each path only includes non-critical activities.*/
      Remove path $p(j)$ from P_{CT};
      Delete critical activities from $p(j)$;
      If $p(j)$ is not empty, then add it to P_{CT};
5. Stop.

In PPCA, the time complexity of Step 2 is $O(n^2)$, whereas the time complexities of Step 3 and Step 4 are less than $O(|P_{CT}| \times n^2)$. Therefore, the time complexity of PPCA is less than $O(|P_{CT}| \times n^2)$.

In Eq. (4), the fact that the start time of an activity is the finish time of its parent (including $p(n)$) indicates that critical activities can be executed serially. They also meet the deadline constraints because the latest completion time of activity n is equal to the deadline δ_n. For non-critical activities, the early start times and finish times are delayed according to precedence relationships. The Current Tree *CT* is still a feasible schedule because the precedence relationships among activities are still preserved, and the workflow completion time satisfies the given deadline. The updated path set only includes those activities that are unassigned activity floats. In the next section, the float allocation and service selection for non-critical activities can be solved according to the Current Tree *CT* and the path set P_{CT}.

5.2.2. Cost optimization for non-critical activities

It is more complex to compute the activity floats and determine the time windows for non-critical activities than for critical activities. To maximize the time intervals of non-critical activities while preserving the precedence relationships, an iterative procedure is presented. Let U denote non-critical activities that are not assigned activity floats. The main steps include:

(i) *Find those activities from set U whose activity floats can be calculated.* All activities with the same immediate successor are called synchronization activities. They can be set to the same finish time without affecting the possible start time of the successors. Thus, once one of them has been assigned an activity float, the allowable finish time for other synchronization activities can be determined. Let σ_r be the activity float of activity, and let W denote the set in which all activities can compute their activity floats. Then set W can be given by

$$W = \left\{ r | \sigma_r = \min_{(r,q) \in E} \{s_q - f_r\}, r \in U, q \in V \setminus U \right\} \tag{5}$$

(ii) *Select the activity with the minimum activity float.* Activity float σ_r is the maximum delay of activity r. The activity with the minimum activity float should be first allocated to satisfy the precedence relations. Let r^* be the activity with the minimum activity float; then the activity r^* holds for

$$\sigma_{r^*} = \min\{\sigma_r | r \in W\} = s_{q^*} - f_{r^*}, \quad \text{for } r^* \in W, q^* \in U \setminus V \tag{6}$$

(iii) *Find the path (or sub-path) that includes activity r^* from P_{CT}.* Let $p(r^*)$ be the path (or sub-path) in P_{CT}, which consists of r^* and its parents. Activity float σ_{r^*} is also the maximum delay of path $p(r^*)$, and it is assigned equally to each activity on $p(r^*)$. Let σ'_{r^*} be an activity float. Then σ'_{r^*} can be computed as follows:

$$\sigma'_{r^*} = \left\{ \frac{\sigma_{r^*}}{|p(r^*)|} \right\} \tag{7}$$

(iv) *Allocate an activity float to activity r^* and determining its time intervals.* The finish time and start time of r^* can be calculated as follows:

$$\begin{cases} f_{r^*} = s_{q^*} \\ s_{r^*} = f_{r^*} - t^*_{r^*} - \sigma'_{r^*} \end{cases} \tag{8}$$

(v) *Build the new Current Tree*. For the successors of activity r^* in the Current Tree, they must delay their start times and finish times with regard to meeting the precedence relations, Add arc (r^*,q^*) to E_{CT}, and delete arc (i,r^*) from E_{CT} (where i is parent of r^* in E_{CT}).

(vi) *Compute the path set P_{CT} of the new Current Tree.* Delete activity r^* from those paths including it. If the path is empty, then it should be removed. If the path is divided into two new paths, they are added to P_{CT}.

(vii) *Delete r^* from U, i.e., $U = U\backslash\{r^*\}$.*
When the set U is empty, the time windows for non-activities are accomplished. However, by analyzing Eqs. (7) and (8), some activity floats may not be assigned fully to those activities on the same path due to complex precedence relationships. In order to take advantage of activity floats, the time windows of non-critical activities can be further adjusted. Thus, the float allocation procedure is described as follows.

Float allocation non-critical activities (FANCA)

1. $U \leftarrow V\backslash p(n)$
2. While ($U \neq \Phi$)
   2.1. Find those activities whose floats can be calculated in set U by Eq. (5), denoted as W;
   2.2. Select the activity with the minimum activity float from W by Eq. (6), denoted as r^*;
   2.3. Find the path (or sub-path), including activity r^* from P_{CT}, and compute its activity float by Eq. (7), denoted as σ'_{r^*};
   2.4. Allocate an activity float to activity r^* and determine its time intervals by Eq. (8);
   2.5. Build the new Current Tree CT;
   2.6. Update the path set P_{CT};
   2.7. $U = U\backslash\{r^*\}$;
3. For $i \in (V\backslash p(n))$ /*Further adjust the time windows of non-critical activities*/
   3.1. $f_{\max} \leftarrow \max\{f_j | j \in pred(i)\}$;
   3.2. If $s_i > f_{\max}$ then $s_i \leftarrow f_{\max}$;
   3.3. $s_{\min} \leftarrow \min\{s_j | j \in succ(i)\}$;
   3.4. If $f_i < s_{\min}$ then $f_i \leftarrow s_{\min}$;
4. Stop.

For FANCA, the computation time is mainly determined by Step 2. The time complexity of Step 2.1 is $O(|U| \times |V\backslash U|)$. The time complexity of Step 2.2 depends on the length of set W, i.e., $O(|W|)$. The set lengths are less than the number of activities. The time complexities of Step 2.3, Step 2.5 and Step 2.7 are less than $O(n)$. It is known that the time complexity of Step 2.7 is less than $O(n^2)$. Therefore, the time complexity of FANCA is less than $O(|U| \times n^2)$.

After determining the time intervals of non-critical activities, each activity can be mapped to the appropriate service within its time window. However, for any one group of sequential tasks, the time intervals can be merged into a single time window, in which its start time is the beginning of the first task, and its finish time is the end of the last task. Thus, the optimal cost solution can also be solved by the dynamic programming method in Section 5.2.1. The service selection for non-critical activities is described by

Service selection for non-critical activities (SSNCA)

1. Search for sequential branches that include non-critical activities, denoted as SB.
2. For $\forall sb_i \in SB$
   2.1. Merge the time windows;
   2.2. Select the appropriate services for sb_i by the dynamic programming method in Section 5.2.1;
3. For other individual non-activity, select the service with the minimum execution cost under its time window by Eq. (3).
4. Stop.

5.3. An illustrative example

The proposed procedure can be illustrated through the example of the workflow application in Fig. 2. The service pool for the example is shown in Table 1. Let the deadline be 90, i.e., $\delta_n = 90$.

For critical activities, the service selections can be determined by the SSCA procedure. The results are listed at the left of Table 2. The earliest completion time for the Critical Path is equal to 83, i.e., the critical activity float is 7. The activity float for critical activities is $a_s = (90 - 83)/(9 - 2) = 1$. The whole deadline is divided into the time windows of critical activities by Eq. (4), which are also listed at the left of Table 2. By the PPCA procedure, Current Tree CT is generated and the path set P_{CT} of CT is updated. The generated Current Tree is the same as the Early Tree in Fig. 3. The path set $P_{CT} = \{p(3) = \{3\}, p(5) = \{5\}, p(8) = \{4,8\}, p(11) = \{11\}, p(14) = \{10,14\}\}$.

For non-critical activities, the time windows are computed step by step. Let parameter $U = \{3,4,5,8,10,11,14\}$; then $V \backslash U = \{1,2,6,7,9,12,13,15,16\}$.

Step 1. Compute the activity floats for activity 3 and 14 using Eq. (4), i.e.$W = \{\{3,4\}|\sigma_3 = 15, \sigma_{14} = 15\}$. They have the same activity float. Select the activity 3 from set W, and it includes path $p(3)$. According to Eq. (7), $\sigma'_3 = 15$. Allocate an activity float 15 to activity 3 by Eq. (8); this gives values of $f_3 = 15$ and $s_3 = 0$. Update the Current Tree CT: Delete arc $(1,3)$ from E_{CT} and add arc $(3,7)$ to E_{CT}. Update path set P_{CT}: path $p(3)$ is deleted, i.e., $P_{CT} = \{p(5) = \{5\}, p(8) = \{4,8\}, p(11) = \{11\}, p(14) = \{10,14\}\}$. Delete activity 3 from set U, then $U = U\backslash\{3\} = \{4,5,8,10,11,14\}$.

Step 2. Compute the activity float for activity 14 using Eq. (4), i.e., $W = \{\{14\}|\sigma_{14} = 15\}$. Only activity 14 includes set W, and it also includes path $p(14)$. According to Eq. (7), $\sigma'_{14} = 15/2 = 7.5$. Allocate an activity float of 7.5 to it via Eq. (8), i.e.$f_{14} = 76$ and $s_{14} = 53.5$. Update the Current Tree CT: Delete arc $(10,14)$ from E_{CT} and add arc $(14,15)$ to E_{CT}. Update path set P_{CT}: Delete path $p(14)$ from it, i.e. $P_{CT} = \{p(5) = \{5\}, p(8) = \{4,8\}, p(11) = \{11\}, p(10) = \{10\}\}$. Delete activity 14 from set U, then $U = U\backslash\{14\} = \{4,5,8,10,11\}$.

Step 3. Compute the activity floats for activity 5, 8, 10 and 11 via Eq. (4); then, $W = \{\{5,8,10,11\}|\sigma_5 = 53.5, \sigma_8 = 44.5, \sigma_{10} = 7.5, \sigma_{11} = 16.5\}$. Activity 10 has the minimum activity float, and it includes path $p(10)$. According to Eq. (7), $\sigma'_{10} = 7.5$. Allocate an activity float of 7.5 to it by Eq. (8), i.e., f_{10}= 53.5 and $s_{10} = 31$. Update the Current Tree CT: Delete arc $(7,10)$ from E_{CT} and add arc $(10,14)$ to E_{CT}. Update path set P_{CT}: Delete path $p(10)$ from it; then, $P_{CT} = \{p(5) = \{5\}, p(8) = \{4,8\}, p(11) = \{11\}\}$. Delete activity 10 from set U, i.e., $U = U\backslash\{10\} = \{4,5,8,11\}$.

Step 4. Compute the activity floats for activity 5, 8 and 11 via Eq. (4), i.e., $W = \{\{5,8,11\}|\sigma_5 = 53.5, \sigma_8 = 44.5, \sigma_{11} = 16.5\}$. Activity 11, including path $p(11)$, has the minimum activity float; then, $\sigma'_{11} = 16.5$ from Eq. (7). Allocate an activity float of 16.5 to it via Eq. (8), giving $f_{11} = 53.5$ and $s_{11} = 31$. Delete arc $(7,11)$ from E_{CT} and add arc $(11,14)$ to E_{CT}; then, the Current Tree CT is updated. Update path set P_{CT}: Delete path $p(11)$ from it, i.e., $P_{CT} = \{p(5) = \{5\}, p(8) = \{4,8\}\}$. Update set U: $U = U\backslash\{11\} = \{4,5,8\}$.

Step 5. Compute the activity floats for activity 5, 8 using Eq. (4), i.e., $W = \{\{5,8\}|\sigma_5 = 53.5, \sigma_8 = 44.5\}$. Activity 8, including path $p(8)$, has the minimum activity float, giving $\sigma'_8 = 44.5/2 = 22.25$. Allocate the activity float of 22.25 to it via Eq. (8), i.e. $f_8 = 53.5$ and $s_8 = 28.25$. Update the Current Tree CT: Delete arc $(4,8)$ from E_{CT} and add arc $(8,14)$ to E_{CT}. Compute path set P_{CT}: Delete sub-path $p(8)$ from it and add sub-path $p(4)$ to it, giving $P_{CT} = \{p(5) = \{5\}, p(4) = \{4\}\}$. Update set U: $U = U\backslash\{8\} = \{5,4\}$.

Step 6. Compute the activity floats for activity 5, 4 using Eq. (4), i.e., $W = \{\{5,4\}|\sigma_5 = 53.5, \sigma_4 = 22.25\}$. Activity 4 has the minimum activity float, which includes path $p(4)$. $\sigma'_4 = 22.25$ by Eq. (7). Allocate an activity float of 22.25 to it via Eq. (8), i.e. $f_4 = 28.25$ and $s_4 = 0$. Update the Current Tree CT: Delete arc $(1,4)$ from E_{CT} and add arc $(4,8)$ to E_{CT}. Compute the path set P_{CT}: Delete path $p(4)$ from it, then $P_{CT} = \{p(5) = \{5\}\}$. Delete activity 4 from set U, i.e., $U = \{5\}$.

Step 7. Compute an activity float for activity 5 by Eq. (4), i.e., $W = \{\{5\}|\sigma_5 = 53.5\}$. Only activity 5 includes Set W. Allocate an activity float of 53.5 to it by Eq. (8), i.e., $f_5 = 53.5$ and $s_2 = 0$. Update the Current Tree E_{CT}: Delete arc $(1,5)$ from E_{CT} and add arc $(5,14)$ to E_{CT}. Compute the path set P_{CT}: Delete path $p(5)$ from it, then $P_{CT} = \Phi$. Delete activity 5 from set U, i.e., $U = \Phi$.

Table 2
Time windows and service selections for all activities in G.

CP	s_i	f_i	S_{ik}	NCP	s_i	f_i	S_{ik}
1	0	0	–	3	0	15	S_{31}
2	0	11	S_{21}	14	53.5	76	$S_{14,2}$
6	11	15	S_{61}	10	31	53.5	$S_{10,2}$
7	15	31	S_{71}	11	31	53.5	$S_{11,1}$
9	31	40	S_{91}	8	28.25	53.5	S_{81}
12	40	50	$S_{12,1}$	4	0	28.25	S_{41}
13	50	76	$S_{13,1}$	5	0	53.5	$S_{5,1}$
15	76	90	$S_{15,1}$				
16	90	90	–				

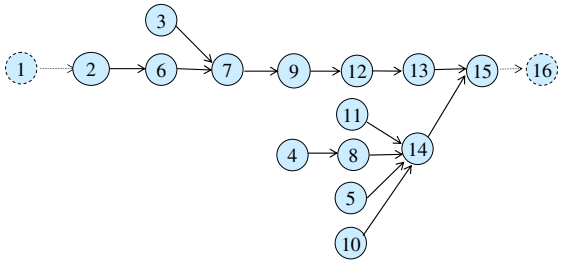


Fig. 4. New spanning tree of G in Fig. 2.

Since set U is empty, the float allocation procedure stops. The new generated Current Tree is shown in Fig. 4. It is still a feasible schedule. The time windows for non-critical activities are listed on the right of Table 2. By running the SSNCA procedure, the appropriate service selections for non-critical activities can be found, which are shown on the right of Table 2.

6. Experimental results

The proposed algorithm is compared with DTL and DBL. The local optimization strategy for serial tasks based on dynamic programming can also be adopted by DBL and DTL. We modified the two leveling algorithms according to SSCA to improve their performance. All these algorithms are coded in Java and performed on a Pentium IV with a 2.93 GHz processor and 512 MB RAM using the operating system Window XP.

Since no standard test instances are available to test the presented heuristics, a DAG graph random generator is developed to generate application DAG with different scales and different structures. The number of activities varies between 20 and 200 with an increment of 20, i.e., $V \in \{20,40,60,80,100,120,140,160,180,200\}$. The maximum out-degree of the DAG ranges from 1 to 3, i.e., $out\_degree \in \{1,2,3\}$. Each workflow instance is given 12 different deadlines, i.e., $\delta_n = T_{\min} + \theta \times (T_{\max} - T_{\min})$, where $T_{\min}$ is the minimum completion time of a workflow instance, and $T_{\max}$ is the maximum completion time of a workflow instance when all tasks select the slowest service for execution. Let $\theta \in \{0.05,0.10,0.15,0.20,0.25,0.30,0.35,0.40,0.45,0.50,0.55,0.60\}$. For each task in DAG, its service pool length is randomly generated from the interval [10,15]. For those services in the same service pool, their execution costs are inversely proportional to their processing times. Each problem class comprises 15 random instances and, in total, 7200 instances are tested.

To compare the effectiveness and efficiency of these heuristics, the experimental results are compared with respect to some measures. Let H denote the heuristic name, and $UB_i(H)$ be the objective function value obtained by algorithm H running on instance i. Assume N is the number of group instances, and $C(H)$ denotes the number of instances for which heuristic H finds the best solutions on the group instances. Let $best_i$ be the best solution that all heuristics can obtain at instance i, and let $t_i(H)$ be the run time that heuristic H runs on instance i. The following measures are given below [20]:

avdev	$\sum_{i=1}^{n}((UB_i(H) - best_i)/best_i)/N$	Average relative deviation from the best solutions on a group of instances
maxdev	$\underset{1\leqslant i\leqslant N}{MAX}\{(UB_i(H) - best_i)/best_i\}$	Maximum relative deviation from the best solutions on a group of instances
OPT (%)	$100 * C(H)/N$	The percent of the number of instances for which a heuristic finds the best solutions on a group of instances
ART (ms)	$\sum_{i-1}^{N} t_i(H)/N$	The average computation time for which a heuristic runs on a group of instances

6.1. Computational results of different instance sizes

The comparison results for the three algorithms running on different instance sizes are depicted in Table 3. With respect to the measures avdev and maxdev, we can see that DET always yields much smaller values than the other two leveling algorithms for any given instance size. Furthermore, it can obtain the best solutions for more than 92% of the instances at any given size, whereas DTL has the same value (equal to 0) and DBL can obtain only a few best solutions. This is because the proposed approach partitions all tasks into different paths based on the Early Tree. Under the given deadline constraint, critical activities can obtain the optimal cost solution based on dynamic programming. For non-critical activities, the activity floats are calculated step by step such that their time intervals are maximized without destroying precedence constraints. The service selection strategy for non-critical activities also implements the local cost optimization. However, the other two leveling algorithms partition the workflow tasks into levels based on their top (bottom) depths, which take into account

Table 3
Computational results depending on different instance sizes.

Tasks	DTL				DBL				DET			
	avdev	maxdev	OPT	ART	avdev	maxdev	OPT	ART	avdev	maxdev	OPT	ART
20	0.385	0.937	0.00	1	0.126	0.376	7.22	2	0.004	0.254	92.78	25
40	0.433	1.050	0.00	2	0.115	0.399	5.42	7	0.003	0.202	94.58	139
60	0.477	1.133	0.00	3	0.122	0.300	2.08	18	0.001	0.079	97.92	411
80	0.488	0.931	0.00	4	0.125	0.301	0.97	45	0.001	0.142	99.03	938
100	0.478	0.960	0.00	6	0.126	0.273	1.25	79	0.000	0.046	98.75	1688
120	0.505	1.139	0.00	8	0.129	0.281	1.25	138	0.000	0.073	98.75	2750
140	0.501	1.045	0.00	9	0.127	0.314	1.81	215	0.000	0.074	98.33	4268
160	0.508	1.076	0.00	11	0.129	0.256	2.50	312	0.000	0.091	97.50	6285
180	0.522	1.203	0.00	14	0.133	0.246	0.69	433	0.000	0.036	99.31	8680
200	0.513	1.065	0.00	16	0.127	0.283	0.97	572	0.000	0.061	99.03	11644
Aver.	0.481	1.203	0.00	7	0.126	0.399	2.42	182	0.001	0.254	97.60	3683

only the parallel and synchronization characteristics of tasks. Some activities cannot obtain more appropriate time intervals. In addition, the dynamic programming method is only applied to groups of serial tasks. Therefore, the deadline division strategy based on the Early Tree is more appropriate than the leveling division policies.

From Table 3, it is also shown that DET decreases the values of the measures avdev and maxdev when the instance size increases, whereas the two leveling heuristics have higher values at any given instance size. For the measure OPT, DTL finds none of the best solutions at any given size. DBL decreases the value, but the proposed approach can find more and more of the best solutions as the size increases. The three algorithms partition tasks into groups according to workflow structure characteristics. When the instance size increases, its structure is more complex. Because the DET algorithm considers not only the serial and parallel properties but also the Critical Path properties of DAG, its advantage is more distinctive than the leveling algorithms as the instance size increases.

As for the average computation time, three heuristics take more computation time when the instance sizes increase. This is because the time complexity of the dynamic programming strategy is dependent on the number of activities. The two leveling algorithms need only a little computation time at any given size. Their average computation time is still less than 1 s when the instance size increases to 200, i.e., they run efficiently, whereas DET requires more computation time as the instance size increases. This is because the dynamic programming strategy for the Critical Path and the iterative procedure for non-critical activities requires considerable computation effort. The average computation time increases to 11.6 s when the instance size is increased to 200, but this time is still acceptable.

6.2. Computational results of different deadlines

Deadline δ_n is the expected latest completion time, which should be greater than or equal to the workflow earliest completion time $C_{p(n)}$. Generally, the longer the deadline, the less cost the workflow expends. In this section, the performance at different deadlines is tested. Table 4 shows the computation results when the parameter θ ranges from 0.05 to 0.6 in an increment of 0.05.

From Table 4, we can see that DET still performs the best on different deadlines. With respect to the measures avdev and maxdev, DET always yields much smaller values than the other two leveling algorithms. Moreover, in all tested instances,

Table 4
Computational results depending on different deadlines.

θ	DTL				DBL				DET			
	avdev	maxdev	OPT	ART	avdev	maxdev	OPT	ART	avdev	maxdev	OPT	ART
0.05	0.459	1.079	0.00	4	0.195	0.206	0.33	32	0.000	0.013	99.67	165
0.10	0.448	1.090	0.00	4	0.122	0.252	0.50	48	0.000	0.026	99.50	418
0.15	0.501	1.181	0.00	5	0.140	0.275	0.83	68	0.000	0.084	99.17	813
0.20	0.498	1.166	0.00	5	0.127	0.268	0.67	91	0.000	0.070	99.33	1320
0.25	0.494	1.128	0.00	6	0.139	0.326	0.33	117	0.000	0.021	99.67	1940
0.30	0.534	1.203	0.00	7	0.134	0.337	1.17	147	0.000	0.088	98.83	2663
0.35	0.475	1.078	0.00	8	0.123	0.277	1.00	180	0.000	0.045	99.00	3488
0.40	0.531	1.118	0.00	8	0.162	0.376	1.17	216	0.000	0.051	98.83	4412
0.45	0.525	1.136	0.00	9	0.125	0.364	2.17	256	0.001	0.112	97.83	5440
0.50	0.473	1.019	0.00	10	0.126	0.309	2.83	297	0.001	0.107	97.17	6577
0.55	0.442	0.928	0.00	11	0.115	0.311	4.50	343	0.002	0.164	95.50	7805
0.60	0.441	0.873	0.00	12	0.102	0.399	13.50	391	0.006	0.254	86.67	9153
Aver.	0.481	1.203	0.00	7	0.126	0.399	2.42	182	0.001	0.254	97.60	3683

DET also finds the best solutions for more than 95% of the instances at different deadlines, i.e., the number of the best solutions achieved by DET is greater than the other two leveling algorithms at any given deadline. On one hand, DET segments the whole deadline into the time windows of tasks based on their dependencies and synchronous constraints, which is adapted for all feasible deadlines. Therefore, the cost curve is smoothly decreasing during the whole valid interval of the deadline. On the other hand, the optimal cost solution for critical activities can be achieved by the dynamic programming method, and the dynamic programming method performs better with longer deadlines. In addition, the iterative procedure manages to maximize the time intervals of all tasks according to the activity float. However, the DTL algorithm divides the deadline by considering the parallel and synchronous starting properties among activities, and the DBL algorithm divides the deadline according to the parallel and synchronous finish characteristics. Some activities cannot obtain more proper time intervals. Furthermore, the whole deadline is segmented into level deadlines that may partially destroy the precedence relationships among tasks. The changed precedence relationships may increase the workflow earliest completion time such that the leveling policies cannot be applied to shorter deadlines. Therefore, DET can achieve better performance than the leveling algorithms within the valid range of the deadline.

From Table 4, it is also shown that both avdev and maxdev increase for the DET heuristic as the deadline extends, but the two leveling algorithms decrease the two measures. As for OPT, DTL cannot find the best solutions (equal to 0). DBL increases the value, whereas the proposed DET heuristic decreases its value as the deadline increases. As described previously, the deadline division strategies based on leveling may change the partial precedence relationships among activities such that the earliest workflow completion time may be longer. In other words, these leveling optimization strategies cannot optimize effectively the total cost for the shorter deadlines. DET considers temporal dependencies, and the deadline division is based on the Critical Path of the Early Tree, which can be applied to all feasible deadlines. When the deadline is shorter, the deadline division is strongly dependent on these temporal constraints such that DET considerably improves the performance of the two leveling algorithms. When the deadline is longer, each task can obtain a larger time window due to the longer total float. All of the tasks can select the appropriate services that optimize the total cost. Therefore, leveling algorithms can obtain a slightly better performance for longer deadlines.

With respect to ART, the computation time for the 3 heuristics increases when the deadline increases. The time complexity for the dynamic programming strategy is also dependent on the time window length allocated to each activity. When the deadline increases, the time windows for the serial tasks and critical activities also become larger. So, the computation time increases. For the two leveling algorithms, they still need a little computation time at any given deadline. The average computation time is still less than 1 s when θ increases to 0.6, i.e., they run efficiently at different deadlines. For the DET algorithm, its average computation time increases to 9.1 s when the deadline parameter is increases to 0.6 and it is still acceptable.

7. Conclusions

This paper tackles the cost optimization for workflow scheduling represented by DAG with a deadline constraint. It is formulated as the discrete time-cost tradeoff problem in project scheduling. Since the optimization problem is generally NP-hard and the Grid environment is highly dynamic, an effective and efficient heuristic should be developed to solve the larger and hard instances. The deadline division strategy is applied to solve the workflow scheduling described by DAG. The heuristic DET is proposed. An early feasible schedule, defined as an Early Tree (ET), is constructed, in which the Critical Path is defined and workflow tasks are grouped. For critical activities, the optimal cost solution under the given deadline constraint can be obtained by the dynamic programming method. The whole deadline is also segmented into time windows in terms of the slack time float. For non-critical activities, an iterative procedure is proposed to maximize the time windows while keeping the precedence constraints among activities. According to the time window allocation, a local cost optimization method is applied to non-critical activities. Extensive computational testing indicates that the proposed approach can achieve better results than the other two leveling algorithms. Moreover, the deadline division strategy adopted by DET can be applied to all feasible deadlines. In terms of the computation time, DET requires a little more computation effort than the other leveling algorithms, but its average run time is less than 12 s when the instance size increases to 200, which is acceptable in practice.

In addition, resource consumption is one of the important evaluation criteria for workflow scheduling. Only two criteria (i.e., cost and the completion time) are considered in this paper. The objective of our future work involves incorporating resource consumption criterion into the objective function.

Acknowledgements

This work was supported by National Natural Science Foundation of China under Grants (No. 60672092, No. 60504029, No. 60873236) and the National High Technology Research and Development Program of China (863 Program) (No. 2008AA04Z103).

References

[1] D. Abramson, R. Buyya, J. Giddy, A computational economy for Grid computing and its implementation in the Nimrod-G resource broker, Future Generation Computing Systems (FGCS) 18 (8) (2002) 1061–1074.

[2] C. Akkan, A. Drexl, A. Kimms, Network decomposition-based benchmark results for the discrete time-cost tradeoff problem, European Journal of Operational Research 165 (2) (2005) 339–358.
[3] R. Al-Ali, A. Hafid, O. Rana, D. Walker, An approach for quality of service adaptation in service-oriented Grids, Concurrency and Computation: Practice and Experience 16 (5) (2004) 401–412.
[4] R. Al-Ali, O. Rana, D. Walker, S. Jha, S. Sohail, G-QoSM: Grid service discovery using QoS properties, Computing and Informatics Journal, Special Issue Grid Computing 21 (4) (2002) 363–382.
[5] J. Blythe et al., Task scheduling strategies for workflow-based applications in Grids, in: Proceedings of IEEE International Symposium on Cluster Computing and the Grid, Cardiff, Wales, United Kingdom, 2005, pp. 759–767.
[6] R. Buyya, D. Abramson, J. Giddy, H. Stockinger, Economic models for resource management and scheduling in Grid computing, Concurrency and Computation: Practice and Experience Journal (Special Issue on Grid Computing Environments) 14 (13–15) (2002) 1507–1542.
[7] R. Buyya, J. Giddy, D. Abramson, A case for economy Grid architecture for service-oriented Grid computing, in: Proceeding of 10th IEEE Internet Heterogeneous Computing Workshop (HCW 2001), San Francisco, CA, IEEE Computer Society, 2001, pp. 776–790.
[8] K. Cooper, A. Dasgupta, K. Kennedy, et al, New Grid scheduling and rescheduling methods in the GrADS Project, in: Proceedings of NSF Next Generation Software Workshop, International Parallel and Distributed Processing Symposium, Los Alamitos, CA, USA, IEEE CS Press, Berlin, 2004, pp. 199–206.
[9] A. Ďogan, F. Ozguner, Scheduling independent tasks with QoS requirements in Grid computing with time-varying resource prices, in: Proceedings of GRID 2002, Lecture Notes in Computer Science, vol. 2536, Springer, 2002, pp. 58–69.
[10] P. De, E.J. Dunne, J.B. Ghosh, C.E. Wells, Complexity of the discrete time-cost tradeoff problem for project networks, Operations Research 45 (2) (1997) 302–306.
[11] P. De, E.J. Dunne, J.B. Ghosh, C.E. Wells, The discrete time–cost tradeoff problem revisited, European Journal of Operational Research 81 (2) (1995) 225–238.
[12] E. Deelman et al, Mapping abstract complex workflows onto Grid environments, Journal of Grid Computing 1 (1) (2003) 25–39.
[13] I. Foster, C. Kesselman, The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufman Publishers, USA, 1999.
[14] I. Foster, C. Kesselman, J.M. Nick, S. Tuecke, Grid service for distributed system integration, IEEE Computer 35 (6) (2002) 37–46.
[15] I. Foster, M. Fidler, A. Roy, V. Sander, L. Winkler, End-to-end quality of service for high-end applications, Computing Communication 27 (14) (2004) 1375–1388.
[16] J. Frey, T. Tannenbaum, et al, Condor-G: a computation management agent for multi-institutional grids, Cluster Computing 5 (2) (2002) 237–246.
[17] K. Golconda, F. Ozguner, A comparison of static QoS-based scheduling heuristics for a meta-task with multiple QoS dimensions in heterogeneous computing, in: Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, New Mexico, IEEE Press, 2004, pp. 106–116.
[18] X. Gu, K. Nahrstedt, A scalable QoS-aware service aggregation model for peer-to-peer computing Grids, in: Proceeding of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC), Piscataway, NJ, USA, 2002, pp. 73–82.
[19] H. Jin, H. Chen, Z. Lu, X. Ning, QoS Optimizing Model and Solving for Composite Service in CGSP Job Manager, Chinese Journal of Computers 28 (4) (2005) 578–588.
[20] R. Klein, Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects, European Journal of Operational Research 127 (4) (2000) 619–638.
[21] C. Li, L. Li, QoS based resource scheduling by computational economy in computational Grid, Information Processing Letters 98 (2006) 119–126.
[22] C. Li, L. Li, Utility-based QoS optimization strategy for multi-criteria scheduling on the Grid, Journal of Parallel and Distributed Computing 67 (2) (2007) 142–153.
[23] M. Lin, Z. Lin, A cost-effective critical path approach for service priority selections in Grid computing economy, Decision Support Systems 42 (3) (2006) 1628–1640.
[24] D.A. Menascé, QoS in Grid computing, IEEE Internet Computing 8 (4) (2004) 85–87.
[25] A. O'Brien, S. Newhouse, J. Darlington, Mapping of scientific workflow within the e-Protein project to distributed resources, in: Proceedings of UK e-Science All Hands Meeting, Nottingham, UK, 2004, pp. 404–409.
[26] Z. Shi, JJ. Dongarra, Scheduling workflow applications on processors with different capabilities, Future Generation Computer Systems 22 (6) (2006) 665–675.
[27] D. Xu, K. Nahrstedt, Finding service paths in a media service proxy network, in: Proceeding of SPIE/ACM Multimedia Computing and Networking Conference (MMCN), California, USA, 2002, pp. 171–185.
[28] J. Yu, R. Buyya, A novel architecture for realizing grid workflow using tuple spaces, in: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA, IEEE CS Press, 2004, pp. 119–128.
[29] J. Yu, R. Buyya, A taxonomy of workflow management systems for grid computing, Journal of Grid Computing 3 (3-4) (2005) 171–201.
[30] J. Yu, R. Buyya, C.K. Tham, Cost-based scheduling of workflow applications on utility Grids, in: Proceedings of the First IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia, 2005, pp. 140–147.
[31] Y. Yuan, X. Li, Q. Wang, Y. Zhang, Bottom Level based heuristic for scheduling workflows in Grids, Chinese Journal of Computers 31 (2) (2008) 283–290.
[32] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, H. Chang, QoS-aware middleware for web services composition, IEEE Transactions on Software Engineering 30 (5) (2004) 311–327.
[33] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, T. Jordan, E. Quigg, J. Dobson, Grid middleware services for virtual data discovery, composition, and integration, in: Proceeding of the Second Workshop on Middleware for Grid Computing, Toronto, Ontario, Canada, 2004, pp. 57–62.