Future Generation Computer Systems 74 (2017) 1–11

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems

Weihong Chen[a,b], Guoqi Xie[a,b,*], Renfa Li[a,b], Yang Bai[a,b], Chunnian Fan[a,c], Keqin Li[a,d]

[a] College of Information Science and Engineering, Hunan University, Changsha, Hunan 410008, China
[b] The National Supercomputing Center in Changsha, Changsha, Hunan 410008, China
[c] Nanjing University of Information Science and Technology, Nanjing, Jiangsu 410008, China
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

HIGHLIGHTS

- We convert the budget constraint of an application into tasks using the budget level.
- We propose the MSLBL algorithm with low-time complexity.
- We validate that MSLBL performs better than existing algorithms under different conditions.
- We propose the algorithm called minimizing the schedule length using the budget level (MSLBL).
- MSLBL can generate less schedule lengths than existing algorithm under different conditions.

ARTICLE INFO

Article history:
Received 22 October 2016
Accepted 28 February 2017
Available online 5 April 2017

Keywords:
Budget constraint
Heterogeneous clouds
Parallel application
Schedule length

ABSTRACT

As the cost-driven public cloud services emerge, budget constraint is one of the primary design issues in large-scale scientific applications executed on heterogeneous cloud computing systems. Minimizing the schedule length while satisfying the budget constraint of an application is one of the most important quality of service requirements for cloud providers. A directed acyclic graph (DAG) can be used to describe an application consisted of multiple tasks with precedence constrains. Previous DAG scheduling methods tried to presuppose the minimum cost assignment for each task to minimize the schedule length of budget constrained applications on heterogeneous cloud computing systems. However, our analysis revealed that the preassignment of tasks with the minimum cost does not necessarily lead to the minimization of the schedule length. In this study, we propose an efficient algorithm of minimizing the schedule length using the budget level (MSLBL) to select processors for satisfying the budget constraint and minimizing the schedule length of an application. Such problem is decomposed into two sub-problems, namely, satisfying the budget constraint and minimizing the schedule length. The first sub-problem is solved by transferring the budget constraint of the application to that of each task, and the second sub-problem is solved by heuristically scheduling each task with low-time complexity. Experimental results on several real parallel applications validate that the proposed MSLBL algorithm can obtain shorter schedule lengths while satisfying the budget constraint of an application than existing methods in various situations.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

1.1. Background

Today's large-scale scientific applications, such as e-commerce, automotive control, and traffic state predication, have drawn a great amount of demand for the design of high performance computing systems [1]. Such applications comprised of many interdependent modules are usually executed in heterogeneous parallel and distributed environments. Computing grids have been used by researchers from various areas of science to execute complex scientific applications [2]. With the emergence of cloud computing and rapid development of cloud infrastructures, more and more scientific computing applications have been migrated to the cloud, on which a pay-as-you-go paradigm is established and on-demand computational services with difference performance and quality of service (QoS) levels can be offered [3]. In this computing model, users pay only for what they use. Accordingly,

* Correspondence to: College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan, 410082, China.
E-mail addresses: paney@126.com (W. Chen), xgqman@hnu.edu.cn (G. Xie), lirenfa@hnu.edu.cn (R. Li), by1990310@qq.com (Y. Bai), fcn@nuist.edu.cn (C. Fan), lik@newpaltz.edu (K. Li).

http://dx.doi.org/10.1016/j.future.2017.03.008
0167-739X/© 2017 Elsevier B.V. All rights reserved.

time and cost become two of the most important factors cared by users. Typically, the execution speed of the powerful resource is directly proportional to the unit price [4]. Thus, the trade-off between time and cost is the key to DAG scheduling. In this paper, we aim at researching the scheduling of the budget constrained application such that its schedule length is minimized.

1.2. Motivation

The scheduling problem of minimizing the schedule length of budget constrained parallel applications has become challenging. In cloud computing systems, service providers and customers are the two types of roles with conflicting requirements by the server-level agreement (SLA) [5]. For service providers, minimizing the schedule length of an application is one of the most important concerns. For customers, the budget constraints of an application is one of the most important QoS requirements. Many studies have been conducted recently to minimize the schedule length while satisfying budget constraints [6–13]. As heterogeneous systems scale up, distributed and parallel applications with precedence-constrained tasks, i.e., large-scale scientific workflows, are represented by directed acyclic graphs (DAGs), in which the nodes represent the tasks and the edges represent the communication messages between tasks [14,15].

The problem of minimizing the schedule length of a budget constrained application with precedence-constrained tasks has been solved recently in a number of studies [4,10]. However, these studies focus on homogeneous cloud system only. The same problem has been studied for heterogeneous cloud computing systems based on the earliest finish time (EFT). Arabnejad et al. [11] presented the heterogeneous budget constrained scheduling (HBCS) algorithm by presupposing tasks with the minimum execution cost to satisfy the budget constraint of the application, then the spare budget of the application was used by prioritized tasks in turn. Although a quantized measurement is used in HBCS, its limitations still exist as follows.

(1) Prioritized tasks select processors with the highest worthiness value by a series of calculations, which results in tasks with higher priority having more chance to use the spare budget of the application. It is unfairness for tasks with lower priority.

(2) Preassigning the minimum execution cost to each task is not always effective with respect to schedule length minimization if there is not enough budget for executing an application. The tasks that have low priority and not enough budget available tend to select the processor with the minimum execution cost, which will lead to longer schedule length.

1.3. Our contributions

In this paper, we focus on the scheduling problem of budget constrained applications on heterogeneous cloud computing systems, and a fair scheduling algorithm with low-time complexity using the budget level is proposed. The objective is to minimize the schedule length of applications under the budget constraint. The contributions of this study are as follows:

(1) The problem of minimizing the schedule length of a budget constrained parallel application in heterogeneous cloud environments is decomposed into two sub-problems, namely, satisfying the budget constraint and minimizing the schedule length. We illustrate the proof of converting the budget constraint of an application into budget constraints of tasks using the budget level.

(2) The algorithm of minimizing the schedule length using the budget level (MSLBL) is proposed, which minimizes the schedule length of budget constrained parallel applications by preassigning tasks with the budget level cost while not violating the precedence constraints between tasks and budget constraint of the application. It has high-performance and low-time complexity.

(3) We do extensively experiments with both Fast Fourier transform and Gaussian elimination parallel applications. Experimental results validate that the proposed MSLBL algorithm can generated less schedule lengths than the state-of-the-art algorithm under different budget constrained and scale conditions.

The rest of this paper is organized as follows. Section 2 reviews related studies. Section 3 presents related models and preliminaries. Section 4 analyzes the existing algorithm and presents the MSLBL algorithm. Section 5 presents the verification of the MSLBL algorithm. Section 6 concludes this study.

2. Related works

Many studies have investigated the DAG scheduling problem in various computing environments. The QoS-aware scheduling considers the optimization of parameters, such as time and cost, using cloud computing to execute workflows. Hu et al. [14] formulated the task scheduling on parallel processors as a DAG scheduling problem, where a heuristic algorithm was proposed to minimize the schedule length (makespan) of a DAG for a bounded number of processors. The author also proved that the optimal makespan can be obtained and a lower bound on the minimum makespan can be determined for a DAG with arbitrary dependency constraints. Heuristic algorithms are widely accepted because DAG scheduling is an NP-complete problem [15]. Scheduling problems are extended to many forms of constraints and environment settings [16–19]. For time-critical parallel applications, the IaaS Cloud Partial Critical Paths (IC-PCP) method for deadline-constrained applications has been proposed on heterogeneous cloud environments [16]. In [20–22], an extensive study on the scheduling algorithms of grid computing was presented toward performance and makespan minimization. However, financial cost is another important parameter in the cloud. For cost-critical parallel applications, cost-aware scheduling algorithms have been proposed for minimizing execution cost or satisfying the budget constraint on heterogeneous systems [1,23]. However, very few papers have targeted heterogeneous cloud computing environment and designs for minimizing the schedule length of budget constrained applications. In [24], a heuristic algorithm of deadline early tree was presented. It minimized the cost of deadline constrained applications without considering the communication time between tasks. In [4], Wu et al. proposed a critical-greedy (CG) algorithm to minimize the end-to-end delay of budget constrained parallel applications. In this work, the CG algorithm defines a global budget level (GBL) parameter and preassigns tasks with the budget-level execution cost. However, this algorithm is for homogeneous cloud environments, where the communication time between tasks is assumed zero, which is not the practice on heterogeneous cloud computing systems.

In addition to single QoS parameter optimization scheduling, the scheduling problem becomes more challenging when two QoS parameters (i.e., time and cost) are considered simultaneously [12,25–27]. Workflow scheduling to satisfy multiple QoS parameters is an attractive area in cloud computing. Malawski et al. [13] proposed DPDS, WA-DPDS, and SPSS algorithms for workflow ensembles on clouds to satisfy budget and deadline constraints, but their aim was to maximize the number of user-prioritized workflows. Arabnejad et al. [12] proposed a deadline-budget constrained scheduling algorithm (DBCS), which aimed to find a feasible schedule within the budget and deadline constraints. In this work, DBCS transfers the deadline and budget constraints of the application to that of each task by defining the CL and DL of each task. The obtained scheduling may or may not satisfy the deadline constraint while satisfying the budget constraint of the application.

In fact, whether the deadline constraint of an application is satisfied or not can be judged by investigating the actual finish time of the exit task in a schedule. On one hand, the scheduling problem of the budget constrained parallel application can be transferred into minimizing the schedule length of the budget constrained application, then the obtained schedule is accepted only if its schedule length is not larger than the deadline constraint of the application. On the other hand, although the quantitative measure is used, DBCS uses the common deadline span of an application for each task, such that no guarantee to ensure that all tasks are completed within the deadline constraint of the application. A comprehensive survey on grid and cloud workflow scheduling was presented in [28–30].

Among all of these previous works, we select an algorithm that are closer to our context. In [11], the HBCS algorithm presupposes tasks with minimum execution costs and assigns processors to tasks based on parameter *worthiness* on heterogeneous systems with the object of satisfying the budget constraint and minimizing the schedule length of applications. As pointed out in Section 1.2, HBCS can be further optimized to obtain the minimum schedule length. In this study, the goal is to propose a fair scheduling algorithm with low time complexity for minimizing the schedule length of budget constrained parallel applications on heterogeneous cloud computing systems.

3. Problem definition

3.1. System model

As illustrated in Fig. 1, the system model for the scheduling problem on cloud environments includes three layers: the task graph, the resource graph, and the cloud infrastructure layers [4, 31]. The task graph layer comprises of tasks with precedence constraints that users submit for executing complex applications. The resources graph layer represents a network of virtual machines (VMs), and the cloud infrastructure layer consists of interconnected physical computer nodes. According to [7], task placement is to find a placement of processors for executing a task, which is similar to the processor allocation. Task scheduling is to determine the placement and starting time of tasks. When time dimension is added to a placement algorithm, the algorithm is extended to a scheduling algorithm and is also regarded as a 3D placement algorithm. This study aims to provide a DAG task scheduling service on a cloud computing system, including the processor allocation and the determination of the start time of tasks on processors.

The targeted computing platform is composed of a set of heterogeneous processors that provide services of different capabilities and costs [10]. Let $P = \{p_1, p_2, \ldots, p_{|P|}\}$ as the processor set, where $|P|$ represents the size of set P. For any set X, this study uses $|X|$ to denote its size. A parallel application running on processors is represented by the DAG $G = \{N, E, C, W\}$. N represents a set of nodes in G, and $n_i \in N$ represents a task with different execution times on different processors. E is a set of communication edges, and each edge $e_{i,j} \in E$ represents a communication message (i.e., transferred data or time) from task n_i to task n_j. Accordingly, C is the set of communication edges, and $c_{i,j}$ represents the communication message (i.e., the time) between n_i and n_j if they are not assigned to the same processor. W is an $|N| \times |P|$ matrix, where $w_{i,k}$ denotes the execution time of task n_i running on processor p_k. $pred(n_i)$ represents the set of immediate predecessors of task n_i, and $succ(n_i)$ represents the set of immediate successors of task n_i.

In a given DAG, the task without a predecessor is the *entry* task denoted as n_{entry}, and the task without any successor is the *exit* task denoted as n_{exit}. If a DAG has multiple n_{entry} or n_{exit} tasks, a dummy entry or exit task with zero-weight dependencies is added to the graph.

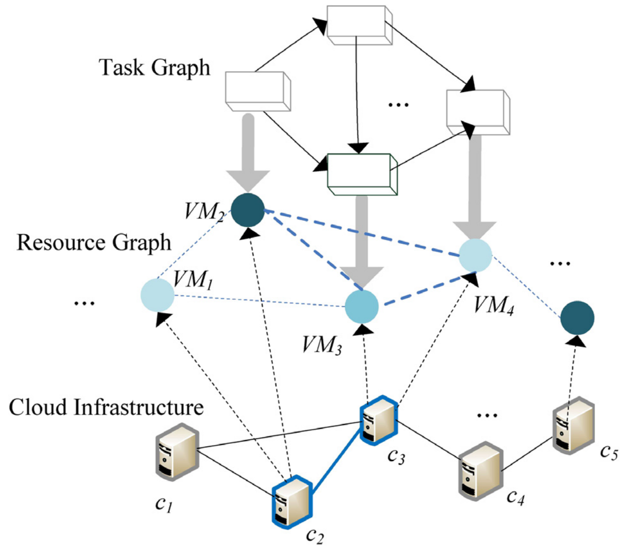

Fig. 1. System model of DAG scheduling service on clouds [4].

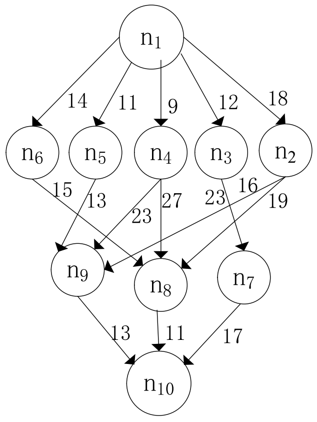

Fig. 2. A DAG-based sample with ten tasks [6].

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

n_i	p_1	p_2	p_3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

Fig. 2 shows a motivating example of a DAG-based parallel application with 10 tasks. Table 1 provides the execution times of tasks on three processors $\{p_1, p_2, p_3\}$. As shown in Fig. 2, the weight 12 of the edge between n_1 and n_3 represents the communication time denoted as $c_{1,3} = 12$ if n_1 and n_3 are not assigned to the same processor. As indicated in Table 1, the execution time of task n_1 on processor p_1 is 14, denoted as $w_{1,1} = 14$. The same task has different execution times on different resources because of the heterogeneity of processors.

3.2. Cost model

The cost model is based on a pay-as-you-go condition, and the users are charged according to the amount of time that they have used processors. Each processor has an individual unit price because processors in the system are completely heterogeneous. We assume that $price_k$ is the unit price of the allocated computing service on processor p_k. All computation and storage services are assumed to be in the same physical region. Hence, communication time $c_{i,j}$ only depends on the amount of data to be transferred between task n_i and task n_j and is independent of the computation service on the processors [21]. The only exception is when both tasks n_i and n_j are executed on the same processor, where $c_{i,j} = 0$. Furthermore, the internal data transfer cost is free in most actual applications, so the data transfer cost is assumed to be zero in our model. Accordingly, we formally define the cost $cost_{i,k}$ of task n_i on processor p_k and total cost $cost(G)$ of a DAG as follows:

$$cost_{i,k} = cost(n_i, p_k) = w_{i,k} \times price_k, \tag{1}$$

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

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 processor $f(i)$, $price_{f(i)}$ is the unit price of processor $p_{f(i)}$.

3.3. Budget constraint

As the execution time of each task on processors is known, the minimum and maximum costs of task n_i, denoted by $cost_{\min}(n_i)$ and $cost_{\max}(n_i)$, respectively, can be obtained by traversing all the processors. For the total cost of the application, G is the sum of that of each task, the minimum and maximum costs of a DAG are:

$$cost_{\min}(G) = \sum_{i=1}^{|N|} cost_{\min}(n_i), \tag{3}$$

$$cost_{\max}(G) = \sum_{i=1}^{|N|} cost_{\max}(n_i), \tag{4}$$

respectively.

Assuming that the budget constraint of an application is $cost_{\rm bud}(G)$, then it should be larger than or equal to $cost_{\min}(n_i)$ and less than or equal to $cost_{\max}(n_i)$; otherwise, $cost_{\rm bud}(G)$ is not always satisfied. Hence, this study assumes that $cost_{\rm bud}(G)$ belongs to the scope of $cost_{\min}(n_i)$ and $cost_{\max}(n_i)$, that is,

$$cost_{\min}(G) \le cost_{\rm bud}(G) \le cost_{\max}(G). \tag{5}$$

3.4. Problem formulation

This study addresses the task scheduling problem of assigning an available processor with a proper price for each task while minimizing the schedule length of the application and ensuring that the total cost of the application does not exceed the budget constraint. The formal description is finding the processor and budget assignments for all tasks with the objective of minimizing the schedule length of the application,

$$SL(G) = AFT(n_{\rm exit}),$$

where $AFT(n_{\rm exit})$ represents the actual finish time (AFT) of the exit task, subject to its budget constraint:

$$cost(G) = \sum_{i=1}^{|N|} cost(n_i, p_{f(i)}) \le cost_{\rm bud}(G). \tag{6}$$

Table 2 shows the mathematical notations used in this paper.

Table 2
Mathematical notations in this paper.

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 immediate predecessors of task n_i
$succ(n_i)$	Set of immediate successors of task n_i
$rank_u$	Upward rank value of tasks
$f(i)$	Index of the processor assigned to task n_i
$price_k$	Unit price of processor p_k
$cost_{i,k}$	$cost(n_i, p_k)$, cost of task n_i on processor p_k,
$cost_{\min}(n_i)$	Minimum cost of executing task n_i
$cost_{\max}(n_i)$	Maximum cost of executing task n_i
$cost_{\min}(G)$	Minimum cost of the application G
$cost_{\max}(G)$	Maximum cost of the application G
$cost_{\rm bud}(G)$	Budget constraint of the application G
$cost(G)$	Total cost of the application G
$SL(G)$	Schedule length of the application G
bl	Budget level of an application
$cost_{\rm bl}(n_i)$	Cost of unassigned task n_i
$cost_{\rm bud}(n_i)$	Budget constraint of task n_i
$\Delta cost(n_i)$	Spare budget of task n_i

Table 3
Upward rank values for tasks in Fig. 2 [6].

n_i	n_1	n_3	n_4	n_2	n_5	n_6	n_9	n_7	n_8	n_{10}
$rank_u(n_i)$	108	80	80	77	69	63	44	43	36	15

3.5. Task prioritization

We first need to set the priority of task assignment before assigning tasks to processors. Similar to [11,12], we employ $rank_u$ of a task as the common task priority standard:

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

where $\overline{w_i}$ is the average execution time of task n_i on processors and calculated as $\overline{w_i} = \sum_{k=1}^{|P|} w_{i,k} / |P|$. It represents the length of the longest path from task n_i to the *exit* node $n_{\rm exit}$, including the communication time between tasks and execution time of tasks. For the *exit* task, $rank_u(n_{\rm exit}) = \overline{w_{\rm exit}}$. Table 3 shows the upward rank values of all tasks in Fig. 2. Note that, as observed from Table 3, the task priority by the decreasing order of $rank_u$ is $\{n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, n_{10}\}$.

4. Minimizing schedule length with budget constraints

In this section, the algorithm for minimizing the schedule length using the budget level (MSLBL) is presented, which aims to find a fair schedule policy for minimizing the schedule length of budget constrained applications with low complexity and high performance on heterogeneous cloud computing systems. Such problem is decomposed into two sub-problems, namely, satisfying the budget constraint and minimizing the schedule length. We first solve these two sub-problems separately, and then present the algorithm by integrating the two sub-problems.

4.1. Existing HBCS algorithm

Before the MSLBL algorithm is presented, we analyze the existing algorithm HBCS. Considering that users consume services based on their QoS requirements and minimizing the schedule length of budget constrained applications is one of the most important concerns, the state-of-the-art HBCS [11] is proposed by presupposing the unassigned tasks with the minimum execution cost while still satisfying the budget constraint of the application. The objective of HBCS is to quantitatively select the processor

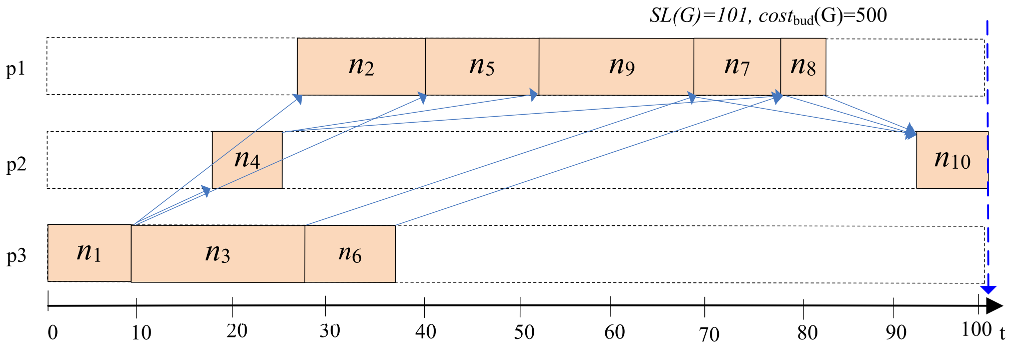

Fig. 3. Scheduling of the application in Fig. 2 with $cost_{\mathrm{bud}}(G) = 500$ using HBCS [11].

Table 4
Unit price of processors in Fig. 2.

p_k	$price_k$
p_1	3
p_2	5
p_3	7

Table 5
Task assignment of the application in Fig. 2 with $cost_{\mathrm{bud}}(G) = 500$ using HBCS [11].

n_i	$\Delta\cos t$	$cost_{\mathrm{bud}}(n_i)$	$AST(n_i)$	$AFT(n_i)$	$f(n_i)$	$cost_{i,f(n_i)}$
1	126	189	0	9	P3	63
3	26	159	9	28	P3	133
4	25	65	18	26	P2	40
2	25	64	27	40	P1	39
5	25	61	40	52	P1	36
6	1	64	28	37	P3	63
9	1	55	52	70	P1	54
7	1	22	70	77	P1	21
8	1	16	77	82	P1	15
10	1	36	94	101	P2	35

$cost(G) = 499 \le cost_{\mathrm{bud}}(G)$, $SL(G) = 101$.

with the highest *worthiness* value for tasks by the descending order of $rank_u$ and minimize the schedule length of the application under the user's specified budget constraint. The core details are explained as follows:

(1) HBCS first invokes the HEFT algorithm on all the processors and obtains $cost_{\mathrm{HEFT}}(G)$ and $cost_{\min}(G)$.

(2) HBCS compares $cost_{\mathrm{HEFT}}(G)$ with $cost_{\mathrm{bud}}(G)$. If $cost_{\mathrm{bud}}(G)$ is larger than or equal to $cost_{\mathrm{HEFT}}(G)$, it returns the schedule map assigned by HEFT.

(3) HBCS preassigns each task with the minimum execution cost and initializes the parameters, then traverses all processors for all tasks by the descending order of $rank_u$. It determines the processor for tasks by calculating the parameters $FT(n_i, p_k)$, $Cost(n_i, p_k)$, $Cost_{Coeff}$, $worthiness(n_i, p_k)$ and RB.

(4) HBCS assigns tasks to the processor with the highest *worthiness* value and returns the schedule map.

For convenience of description, we define the spare budget to observe the budget assignment of tasks in applications on heterogeneous cloud computing systems.

Definition 1 (*Spare Budget*)**.** Spare budget is defined as the difference between the budget constraint of a task and its preassigned cost, as shown in Eq. (7):

$$\Delta cost(n_i) = cost_{\mathrm{bud}}(n_i) - cost_{\mathrm{pre}}(n_i), \tag{7}$$

where $cost_{\mathrm{bud}}(n_i)$ is the budget constrained cost of task n_i, $cost_{\mathrm{pre}}(n_i)$ is the preassigned cost of the unassigned task n_i. Initially, the spare budget of application G is the difference between $cost_{\mathrm{bud}}(G)$ and $cost_{\mathrm{pre}}(G)$.

We illustrate the motivating parallel application to explain the HBCS algorithm. We set the budget constraint of G as $cost_{\mathrm{bud}}(G) = 500$. We assume that the unit price of processors in Fig. 2 is known, as shown in Table 4. We compute the minimum cost value of the application as $cost_{\min}(G) = 353$ using Eq. (6). Table 5 shows the task assignment of the parallel application in Fig. 2 using HBCS, where all tasks satisfy their individual budget constraints. The actual execution cost of the application is $cost(G) = 499$, and its schedule length is $SL(G) = 101$. Fig. 3 also shows the scheduling of the parallel application G in Fig. 2 with $cost_{\mathrm{bud}}(G) = 500$ using HBCS. Note that the arrows in Fig. 3 represent the generated communication times between tasks.

This example verifies that the use of HBCS can ensure that the actual cost of the application does not exceed the user given budget constraint, that is, $cost(G) \le cost_{\mathrm{bud}}(G)$. However, as shown in Table 5, the task with higher priority has more spare budget than those with low priorities. The potential reason for this is that HBCS presupposes unassigned tasks in the application with the minimum execution cost and the spare budget of the application is shared by the former assigned tasks that have more opportunities. Thus, the task with low priority tends to select processors with minimum costs but long execution times, which may result in a longer schedule length of the application. As shown in Table 5, tasks n_6 to n_{10} have few spare budget available. It is unfair for tasks with low priority. Thus, the HBCS algorithm should be further optimized.

4.2. Satisfying budget constraint

Assuming that the task to be assigned is $n_{s(j)}$, where $s(j)$ represents the jth assigned task, then $\{n_{s(1)}, n_{s(2)}, \ldots, n_{s(j-1)}\}$ represents the task set where the tasks have been assigned, and $\{n_{s(j+1)}, n_{s(j+2)}, \ldots, n_{s(|N|)}\}$ represents the task set where the tasks have not been assigned. Initially, all tasks of the application are unassigned. To ensure fairness for all tasks as much as possible, we first provide the definition of the global level [4].

Definition 2 (*Budget Level*)**.** Budget level is defined as the ratio of the difference between $cost_{\mathrm{bud}}(G)$ and the minimum execution cost of the application to the difference between the maximum and minimum execution costs of application G, as shown in Eq. (8):

$$bl = \frac{cost_{\mathrm{bud}}(G) - cost_{\min}(G)}{cost_{\max}(G) - cost_{\min}(G)}. \tag{8}$$

According to Eqs. (5) and (8), we have $bl \in [0, 1]$. Note that, the original bl is only applied to homogeneous cloud computing systems [4]. In this study, we make bl be suitable for heterogeneous systems by transferring the budget constraint of an application into budget constraints of tasks and complete scheduling based on EFT. The workflow model in [4] does not consider the communication time between tasks. However, we consider both the precedence constraint and the communication time between tasks in our application model. Last but not least, we give the proof of converting the budget constraint of an application into budget constraints of tasks using the budget level.

Then, we preassign unassigned tasks with a budget level cost described as:

$$cost_{\mathrm{bl}}(n_{s(j)}) = cost_{\min}(n_{s(j)}) + (cost_{\max}(n_{s(j)}) - cost_{\min}(n_{s(j)})) \times bl. \tag{9}$$

Correspondingly, $cost_{\mathrm{pre}}(n_{s(j)})$ in Eq. (7) is equal to $cost_{\mathrm{bl}}(n_{s(j)})$. According to Eqs. (8) and (9):

$$cost_{\min}(n_{s(j)}) \le cost_{\mathrm{pre}}(n_{s(j)}) \le cost_{\max}(n_{s(j)}). \tag{10}$$

Hence, when assigning $n_{s(j)}$, the total cost of the application G is calculated as:

$$cost_{n_{s(j)}}(G) = \sum_{i=1}^{j-1} cost_{s(i),f(s(i))} + c_{s(j),p_k} + \sum_{i=j+1}^{|N|} cost_{bl}(n_{s(i)}).$$

For any task $n_{s(j)}$, the actual execution cost $cos(G)$ should be less than or equal to $cost_{\mathrm{bud}}(G)$ only if $cost_{n_{s(j)}}(G) \le \cos t_{\mathrm{bud}}(G)$. Its proof is provided in the following.

Theorem 1. *Each task in the parallel application G can always find an assignment of processors to satisfy:*

$$cost_{n_{s(j)}}(G) = \sum_{i=1}^{j-1} cost_{s(i),f(s(i))} + cost_{s(j),p_k} + \sum_{i=j+1}^{|N|} cost_{\mathrm{bl}}(n_{s(i)}) \le cost_{\mathrm{bud}}(G). \tag{11}$$

Proof. We use the mathematical induction to prove it.

First, for the entry task $n_1 = n_{s(1)}$, all tasks are not assigned to processors:

$$cost_{n_{s(1)}}(G) = cost_{s(1),p_k} + \sum_{i=2}^{|N|} cost_{\mathrm{bl}}(n_{s(i)}) \le cost_{\mathrm{bud}}(G). \tag{12}$$

According to Eqs. (8), (9) and (12), we have

$$\begin{aligned} cost_{n_{s(1)}} &= cost_{s(1),p_k} + \sum_{i=2}^{|N|} cost_{\mathrm{bl}}(n_{s(i)}) \\ &= cost_{s(1),p_k} + \sum_{i=1}^{|N|} cost_{\mathrm{bl}}(n_{s(i)}) - cost_{\mathrm{bl}}(n_{s(1)}) \\ &= cost_{s(1),p_k} + \sum_{i=1}^{|N|} cost_{\min}(n_{s(i)}) \\ &\quad + \left[\sum_{i=1}^{|N|} cost_{\max}(n_{s(i)}) - \sum_{i=1}^{|N|} cost_{\min}(n_{s(i)})\right] \\ &\quad \times bl - cost_{\mathrm{bl}}(n_{s(1)}) \\ &= cost_{s(1),p_k} + \cos t_{\mathrm{bud}}(G) - cost_{\mathrm{bl}}(n_{s(1)}) \end{aligned} \tag{13}$$

Namely, Eq. (13) should be satisfied

$$cost_{s(1),p_k} + \cos t_{\mathrm{bud}}(G) - cost_{\mathrm{bl}}(n_{s(1)}) \le \cos t_{\mathrm{bud}}(G). \tag{14}$$

According to Eq. (10), $cost_{\mathrm{bl}}(n_{s(1)})$ is larger than or equal to $cost_{\min}(n_{s(1)})$, that is, the processor with the minimum execution cost can be assigned to task $n_{s(1)}$ at the least. Hence, $n_{s(1)}$ can find an assigned resource to satisfy Eq. (14), that is, Eq. (11) is satisfied for $n_{s(1)}$.

Second, we assume that the jth task $n_{s(j)}$ can find an assigned processor $p_{f(s(j))}$ to satisfy $cost_{\mathrm{bud}}(G)$, and we have:

$$cost_{n_{s(j)}}(G) = \sum_{i=1}^{j-1} cost_{s(i),f(s(i))} + cost_{s(j),f(s(j))} + \sum_{i=j+1}^{|N|} cost_{\mathrm{bl}}(n_{s(i)}) \le cost_{\mathrm{bud}}(G). \tag{15}$$

such that,

$$cost_{n_{s(j)}}(G) = \sum_{i=1}^{j} cost_{s(i),f(s(i))} + \sum_{i=j+1}^{|N|} cost_{\mathrm{bl}}(n_{s(i)}) \le cost_{\mathrm{bud}}(G).$$

Hence,

$$\sum_{i=1}^{j} cost_{s(i),f(s(i))} \le cost_{\mathrm{bud}}(G) - \sum_{i=j+1}^{|N|} cost_{\mathrm{bl}}(n_{s(i)}). \tag{16}$$

Finally, for the $(j+1)$th task, the execution cost of an application is

$$cost_{n_{s(j+1)}}(G) = \sum_{i=1}^{j} cost_{s(i),f(s(i))} + cost_{s(j+1),f(s(j+1))} + \sum_{i=j+2}^{|N|} cost_{\mathrm{bl}}(n_{s(i)}) \le cost_{\mathrm{bud}}(G). \tag{17}$$

Known as Eqs. (13) and (14), we have:

$$\begin{aligned} cost_{n_{s(j+1)}}(G) &\le cost_{\mathrm{bud}}(G) - \sum_{i=j+1}^{|N|} cost_{\mathrm{bl}}(n_{s(i)}) \\ &\quad + cost_{s(j+1),f(s(j+1))} + \sum_{i=j+2}^{|N|} cost_{\mathrm{bl}}(n_{s(i)}) \\ &= cost_{\mathrm{bud}}(G) - cost_{\mathrm{bl}}(n_{s(j+1)}) \\ &\quad + cost_{s(j+1),f(s(j+1))}. \end{aligned} \tag{18}$$

When $n_{s(j+1)}$ is assigned a cost in range of $cost_{\min}(n_{s(j+1)})$ to $cost_{\mathrm{bl}}(n_{s(j+1)})$, we have:

$$cost_{n_{s(j+1)}}(G) \le cost_{\mathrm{bud}}(G). \tag{19}$$

According to Eq. (10), $n_{s(j+1)}$ can at least be assigned to the processor with the minimum cost, that is, $n_{s(j+1)}$ can also find an assigned processor to satisfy $cost_{\mathrm{bud}}(G)$.

When all tasks can find individual assigned processors to satisfy $cost_{\mathrm{bud}}(G)$, Theorem 1 is satisfied. This completes the proof.

4.3. Minimizing schedule length

HEFT is a well-known precedence-constrained task scheduling, which is based on the DAG model to reduce schedule length to a minimum combined with low complexity and high performance in heterogeneous systems [6,22]. Task prioritization is based on upward rank value ($rank_u$) and task assignment is based on the earliest finish time.

$EST\left(n_j, p_k\right)$ and $EFT\left(n_j, p_k\right)$ denote the earliest start time (EST) and EFT of task n_j on processor p_k, respectively, defined as:

$$EST(n_j, p_k) = \max\{T_{avail(p_k)}, \max_{n_i \in pred(n_j)} \{AFT(n_i) + c_{i,j}\}\}, \tag{20}$$

$$EFT(n_j, p_k) = EST(n_j, p_k) + w_{j,k}, \tag{21}$$

where $T_{avail(p_k)}$ is the earliest time at which the processor p_k is ready for task execution. Communication time $c_{i,j}$ is zero if n_j and its predecessor n_i are assigned to the same resource. For the entry task, $EST(n_{\text{entry}}, p_k) = 0$. $AST(n_i)$ and $AFT(n_i)$ are the actual start time and the actual finish time of task n_i, respectively.

To select the best suitable processor, the cost and time trade-off is evaluated. Typically, the faster the speed of processors is, the higher the execution cost of tasks is. However, this is not always the case on heterogeneous computing systems. After transferring the budget constraint of applications to that of each task, tasks are assigned to processors with the minimum EFT by using the insertion-based scheduling strategy while satisfying the budget constraint of tasks.

4.4. Proposed MSLBL algorithm

We first give the budget constraint of each task before we propose the algorithm. According to Eq. (11), we have:

$$cost_{s(j),f(s(j))} \leq cost_{\text{bud}}(G) - \sum_{i=1}^{j-1} cost_{s(i),f(s(i))} - \sum_{i=j+1}^{|N|} cost_{\text{bl}}(n_{s(i)}).$$

Hence, let the budget constraint of task $n_{s(j)}$ be

$$cost_{\text{bud}}(n_{s(j)}) = cost_{\text{bud}}(G) - \sum_{i=1}^{j-1} cost_{s(i),f(s(i))} - \sum_{i=j+1}^{|N|} cost_{\text{bl}}(n_{s(i)}). \quad (22)$$

Then, the budget constraint of the application can be transferred to that of each task. That is, we just let $n_{s(j)}$ satisfy the following constraint:

$$cost_{s(j),f(s(j))} \leq cost_{\text{bud}}(n_{s(j)}).$$

Hence, when assigning the task $n_{s(j)}$, we can directly consider the budget constraint $cost_{\text{bud}}(n_{s(j)})$ of $n_{s(j)}$ and do not have to be concerned about the budget constraint $cost_{\text{bud}}(G)$ of the application G. In this way, a low time complexity heuristic algorithm can be obtained.

Inspired by the above analysis, we propose the MSLBL algorithm to minimize the schedule length while still satisfying the budget constraint of the application. The steps of MSLBL are described in Algorithm 1.

The main idea of MSLBL is that the budget constraint of the application is transferred to that of each task to change the budget constraint of each task by using the budget level. The algorithm assigns processors to tasks by the descending order of $rank_u$ values, and each task select the processor with the minimum EFT while satisfying its individual budget-level constraint. The core details are explained as follows:

(1) In Lines 1–6, the minimum cost value of G and budget level cost of tasks are computed.

(2) In Lines 7–9, the possibility of finding a schedule under the user provided budget constraint is verified.

(3) In Lines 10–23, the algorithm starts to map the tasks of the application to processors. At each step, the task with the highest priority among the unscheduled tasks is selected as the current task $n_{s(j)}$, and its budget constraint is set. Next, for task $n_{s(j)}$, the processor that satisfies its budget level constraint and has the minimum EFT is selected. After assigning the processor to $n_{s(j)}$, the spare budget $\Delta\cos t$ is updated using Eq. (7) and the for loop is broken.

(4) In Lines 24–25, the actual execution cost and final schedule length $SL(G)$ are computed.

In terms of time complexity, MSLBL requires the computation of $rank_u$ and the minimum and maximum costs of the application that has complexity $O(|N| \times |P|)$. In the processor selection phase for the current task, the complexity is $O(|N| \times |P|)$ for computing EFT, and $O(|P|)$ for selecting the resource with the minimum EFT, satisfying its budget constraint. The total time is $O(|N| \times |P| + |N| \cdot (O(|N| \times |P|) + |P|$, where time complexity is of the order $O(|N|^2 \times |P|)$.

Algorithm 1 The MSLBL Algorithm

```
Input: $G = \{N, E, C, W\}$, for $\forall j, p_j \in P$, $price_j$, $cost_{\text{bud}}(G)$
Output: $SL(G)$, $cost(G)$
1: Sort the tasks in a list dlist by descending order of $rank_u$ values;
2: for ($\forall i, n_i \in N$) do
3:     Compute $cost_{\min}(n_i)$ and $cost_{\max}(n_i)$.
4: end for
5: Compute $cost_{\min}(G)$ and $cost_{\max}(G)$ using Eqs. (3) and (4);
6: Compute $bl$, $cost_{\text{bl}}(n_i)$ and $\Delta cost(n_i)$ using Eqs. (7), (8) and (9), respectively;
7: if ($cost_{\text{bud}}(G) \leq cost_{\min}(G)$) then
8:     return 0;
9: end if
10: while (there are tasks in dlist) do
11:     $n_{s(j)}$ = dlist.out();
12:     Compute $cost_{\text{bud}}(n_{s(j)})$ using Eq. (22);
13:     for ($\forall k, p_k \in P$) do
14:         Compute $cost_{s(j),k}$ and $EFT(n_{s(j)}, p_k)$ using Eq. (21);
15:     end for
16:     for ($k = 1; i \leq |P|; k++$) do
17:         if ($cost_{s(j),k} \leq cost_{\text{bud}}(n_{s(j)})$) then
18:             Select the processor with the minimum EFT for task $n_{s(j)}$;
19:             Update $\Delta cost$ using Eq. (7);
20:             AFT($n_{s(j)}$) = $EFT(n_{s(j)}, p_k)$;
21:         end if
22:     end for
23: end while
24: Compute the actual cost $cost(G)$ using Eq. (2);
25: Compute $SL(G) = AFT(n_{exit})$;
26: return $SL(G)$, $cost(G)$.
```

Table 6
Task assignment of the application in Fig. 2 with $cost_{\text{bud}}(G) = 500$ using MSLBL.

n_i	$\Delta\cos t$	$cost_{\text{bud}}(n_i)$	$AST(n_i)$	$AFT(n_i)$	$f(n_i)$	$cost_{i,f(n_i)}$
1	0	50	0	14	P1	42
3	8	63	14	25	P1	33
4	30	86	23	31	P2	40
2	46	104	25	38	P1	39
5	65	70	25	35	P3	70
6	0	80	35	44	P3	63
9	17	95	54	66	P2	60
7	35	69	38	45	P1	21
8	48	80	59	64	P1	15
10	65	112	75	82	P2	35

$cost(G) = 418 \leq cost_{\text{bud}}(G)$, $SL(G) = 82$.

4.5. Example of the MSLBL algorithm

We present an motivating example to show the results using the MSLBL algorithm. We consider the condition $cost_{\text{bud}}(G) = 500$ for the application in Fig. 2, and the unit price of processors is shown in Table 4. Table 6 shows the task assignment of the parallel application in Fig. 2 with $cost_{\text{bud}}(G) = 500$ using MSLBL. The total cost of the application is 418 and its schedule length is 82, which is less than that using the HBCS algorithm. Fig. 4 shows the scheduling of the parallel application G in Fig. 2 with $cost_{\text{bud}}(G) = 500$ using MSLBL.

5. Experimental results and discussion

This section presents performance comparisons of the MSLBL algorithm with HBCS [11] and DBCS [12] algorithms because they have the similar application models.

5.1. Experimental metrics

We resort to simulation method to verify our algorithms. We implement a simulator in Java language. All simulative experiments are conducted on a PC platform with an Intel Core i5 2.60 GHz CPU and 4 GB memory. The simulated heterogeneous cloud computing system contains 128 processors with different computing abilities and unit prices, where the types and the prices of processors are based on the Amazon Elastic Compute Cloud (EC2) environment [32]. The application and processor parameters

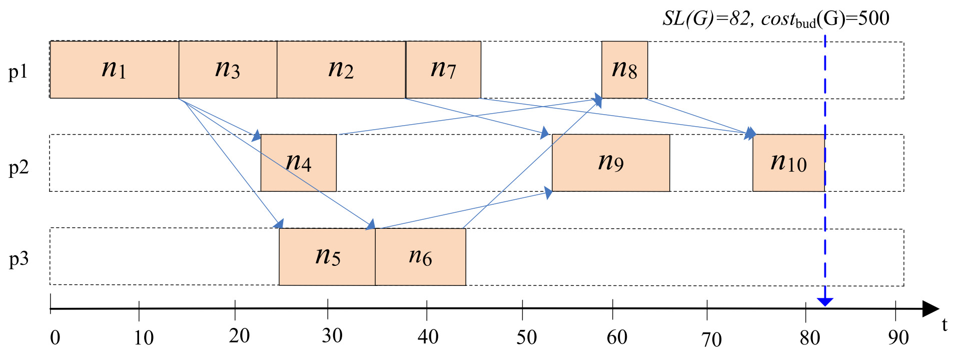

Fig. 4. Scheduling of the application in Fig. 2 with $cost_{\text{bud}}(G) = 500$ using MSLBL.

Table 7
Actual cost and schedule length of Fast Fourier transform applications with budget constraints for varying number of tasks.

$\|N\|$	$cost_{\min}(G)$	$cost_{\text{bud}}(G)$	HBCS [11]		MSLBL	
			$cost(G)$	$SL(G)$	$cost(G)$	$SL(G)$
96	40	48	48	796	47	739
224	76	91	91	1 302	90	984
512	221	265	265	13 945	265	9 967
1152	463	556	556	25 637	556	15 222
2560	770	924	924	31 991	924	22 401

Table 8
Actual cost of the Fast Fourier transform application with $\rho = 256$ (i.e.,$|N| = 2560$) for varying budget constraints.

$\|N\|$	$cost_{\min}(G)$	$cost_{\text{bud}}(G)$	HBCS [11]		MSLBL	
			$cost(G)$	$SL(G)$	$cost(G)$	$SL(G)$
2560	923	1015	1015	48 321	1015	35 134
2560	923	1107	1107	48 319	1107	2 9178
2560	923	1200	1200	48 292	1200	25 428
2560	923	1292	1292	48 235	1292	21 404
2560	923	1384	1384	48 235	1384	19 794

are: $0.01\ \$/\text{h} \leq price_k \leq 1\ \$/\text{h}$, $0.01\ \text{h} \leq w_{i,k} \leq 128\ \text{h}$, $0.01\ \text{h} \leq c_{i,j} \leq 30\ \text{h}$. The number of tasks can be dynamically set and varies between 10 and 2627 in our experiments. The average bandwidth between the computation services is set to 20 MBps which is the approximate average bandwidth of the computation services (EC2) in Amazon [5]. The number of tasks executed on each processor is determined by the scheduling algorithm.

The performance metrics selected for comparison are the actual cost $cost(G)$, the final schedule length $SL(G)$ of budget constrained applications, and the acceptance ratio (AR) of budget and deadline constrained applications. The AR is expressed by Eq. (23):

$$AR = \frac{\text{number of acceptance}}{\text{total number of experiments}}. \tag{23}$$

To verify the effectiveness of the scheduling algorithms, real applications with precedence constrained tasks are widely used in some high-performance computing, such as Fast Fourier transform and Gaussian elimination [6]. We use these two types of real parallel applications under the condition of the budget and budget-deadline constraints to observe the results.

5.2. Fast Fourier transform parallel applications

A new parameter ρ is used as the size for the Fast Fourier transform application, and the number of tasks is $|N| = 2 \times \rho - 1 + \rho \times \log_2 \rho$, where $\rho = 2^y$ for some integer y. Fig. 5 shows an example of the Fast Fourier transform parallel applications with $\rho = 8$. Note that ρ *exit* tasks exist in this application with size ρ. To adapt the application of this study, we create a dummy *exit* task with zero execution time, which connects these ρ *exit* tasks with zero communication time.

5.2.1. Varying number of tasks

This experiment is conducted to compare the actual cost and final schedule lengths of Fast Fourier transform parallel applications for varying number of tasks. We limit $cost_{\text{bud}}(G)$ as $cost_{\min}(G) \times 1.2$. The number of tasks is changed from 96 (small scale) to 2560 (large scale) when ρ is changed from 16 to 256.

As shown in Table 7, the budget constraints and actual costs of the applications increase gradually with the number of tasks. At the same time, the actual cost of applications in different scales using HBCS and MSLBL satisfy the budget constraints. Such results verify that these algorithms can satisfy the budget constraints of applications in practice.

In addition to satisfying the budget constraints of the applications, an exciting phenomenon is that MSLBL can generate shorter schedule length than HBCS in different scale parallel applications. Furthermore, the difference becomes more and more obvious with the increase of the number of tasks. For example, when $|N| = 2560$, the schedule length using MSLBL is 22401, which is 70% of 31 991 that used HBCS. Such results indicate that MSLBL is very suitable for minimizing schedule length of budget constrained parallel applications and is useful for different scale parallel applications.

5.2.2. Varying budget constraints

To observe the performance in different scales of applications, this experiment is conducted to compare the actual cost and final schedule length of Fast Fourier transform parallel applications for varying budget constraints. We limit the size of the application as ρ=256 (i.e.,$|N| = 2560$). $cost_{\text{bud}}(G)$ is changed from $cost_{\min}(G) \times 1.1$ to $cost_{\min}(G) \times 1.5$.

Table 8 shows that the actual costs of applications using HBCS and MSLBL can satisfy the budget constraints in all cases. In addition, the schedule length using MSLBL is less than those of the HBCS algorithm. Furthermore, the difference in the schedule lengths using the two algorithms becomes more and more obvious with the increasing budget constraints of the application. For example, when $cost_{\text{bud}}(G)$ is $cost_{\min}(G) \times 1.1$, the schedule lengths using MSLBL is 35 134, 73% of that using HBCS, whereas that using MSLBL is 19 794 and 41% of that using HBCS when $cost_{\text{bud}}(G)$ is $cost_{\min}(G) \times 1.5$ in this experiment.

5.3. Gaussian elimination parallel applications

To verify the performance further, this experiment uses another important real parallel application (i.e., Gaussian elimination) as experimental object. A new parameter ρ is used as the matrix size of the Gaussian elimination application, and the total number of tasks is $|N| = \frac{\rho^2+\rho-2}{2}$ [6]. Fig. 6 shows an example of the Gaussian elimination parallel application with $\rho = 5$.

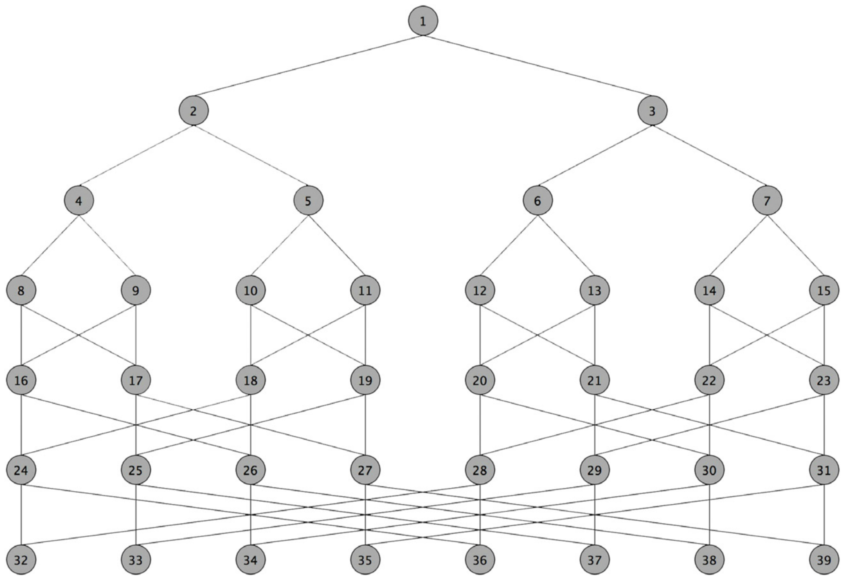

Fig. 5. Example of fast Fourier transform parallel applications with $\rho = 8$.

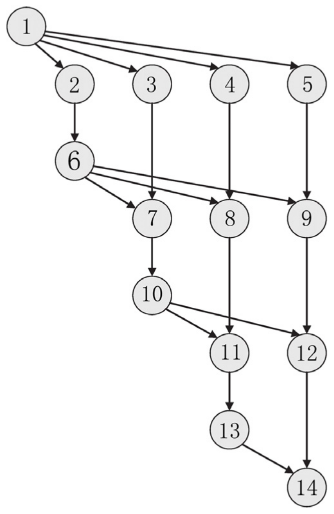

Fig. 6. Example of the Gaussian elimination parallel application with $\rho = 5$.

Table 9
Actual cost and schedule length of Gaussian elimination applications with budget constraints for varying numbers of tasks.

$\|N\|$	$cost_{\min}(G)$	$cost_{\text{bud}}(G)$	HBCS [11]		MSLBL	
			$cost(G)$	$SL(G)$	$cost(G)$	$SL(G)$
77	25	30	30	1 506	30	1 506
299	168	201	201	3 520	201	3 341
665	230	275	275	12 977	275	8 797
1175	439	527	527	21 436	526	16 409
1829	629	755	754	33 903	754	25 076
2627	2142	2570	2570	35 460	2569	22 389

5.3.1. *Varying number of tasks*

To observe the performance in different scales of real applications further, this experiment is conducted to compare the actual costs and final schedule length of Gaussian elimination parallel applications for varying number of tasks. We limit $cost_{\text{bud}}(G)$ as $cost_{\min}(G) \times 1.2$. The number of tasks is changed from 77 (small scale) to 2627 (large scale) when ρ is changed from 12 to 72, which are approximately equal to the numbers of the Fast Fourier transform application in experiment 5.2.1.

Table 10
Actual cost and schedule length of Gaussian elimination parallel applications with $\rho = 72$ (i.e., $|N| = 2627$) for varying budget constraints.

$\|N\|$	$cost_{\min}(G)$	$cost_{\text{bud}}(G)$	HBCS [11]		MSLBL	
			$cost(G)$	$SL(G)$	$cost(G)$	$SL(G)$
2627	1026	1129	1129	61 297	1129	50 800
2627	1026	1232	1232	61 359	1231	46 711
2627	1026	1334	1334	61 096	1334	44 497
2627	1026	1437	1437	61 058	1436	40 957
2627	1026	1540	1540	60 819	1540	38 934

Table 9 shows the results of the Gaussian elimination applications for varying numbers of tasks using two algorithms. The actual cost and schedule length of applications increase with the increase of the number of tasks. However, the schedule length in Gaussian elimination parallel applications is longer than that in Fast Fourier transform parallel applications. Such results indicate that the Fast Fourier transform application has better parasitism than the Gaussian elimination application in the structure, and can generate shorter schedule length.

Similar to the results of Experiment 5.2.1, the actual costs of applications using HBCS and MSLBL can still satisfy and are close to the budget constraints of Gaussian elimination applications in different scales. The schedule length using MSLBL is shorter than that using HBCS in large-scale applications. For example, when the task number exceeds 600, the schedule lengths using MSLBL is 22 389, while that using HBCS are 35 460. The former is merely 63.1% of the latter. Such results indicate that MSLBL can satisfy the budget constraints with the minimum schedule length and generate shorter schedule lengths than HBCS for large-scale Gaussian elimination applications.

5.3.2. *Varying budget constraints*

This experiment is conducted to compare the actual cost and the final schedule length of Gaussian elimination parallel applications for varying budget constraints. We limit the size of the application as $\rho = 72$ (i.e.,$|N| = 2627$), which is approximately equal to the number of tasks of the Fast Fourier transform parallel application in Experiment 5.2.2. $cost_{\text{bud}}(G)$ is changed from $cost_{\min}(G) \times 1.1$ to $cost_{\min}(G) \times 1.5$.

Table 10 shows the results of the Gaussian elimination parallel application for varying budget constraints using the two algorithms. Similar to the results of Table 8 for Experiment 5.2.2,

Table 11
Acceptance ratio of the Fast Fourier transform applications with budget-deadline constraints for varying numbers of tasks.

$\lvert N\rvert$	HBCS [11]	DBCS[12]	MSLBL
96	71%	73%	87%
224	66%	67%	82%
512	36%	36%	64%
1152	10%	10%	37%
2560	0%	0%	15%

Table 12
Acceptance ratio of the Gaussian elimination parallel applications with budget-deadline constraints for varying numbers of tasks.

$\lvert N\rvert$	HBCS [11]	DBCS [12]	MSLBL
77	77%	88%	89%
299	84%	84%	89%
665	77%	77%	81%
1175	62%	62%	83%
1829	60%	60%	88%
2627	47%	47%	73%

the schedule length using MSLBL is shorter than that using HBCS in the same scale levels. When $cost_{\text{bud}}(G)$ is $cost_{\min}(G) \times 1.5$, the schedule lengths using HBCS and MSLBL are 60 819 and 38 934, respectively. The latter is 64% of the former. The actual costs using two algorithms are within the budget constraints in all cases. The overall trend of both Gaussian elimination and Fast Fourier transform applications in the same scale is similar. That is, MSLBL has the shorter schedule length than HBCS for budget constrained parallel applications. Such results indicate that MSLBL is more effective in different types of parallel applications than the state-of-the-art algorithm.

5.4. Verification of budget-deadline constrained scheduling

The proposed approach is verified further by experiments of budget-deadline constrained parallel applications. For the problem of budget and deadline constrained scheduling, a scheduling can be accepted only if its actual cost and schedule length of the application can satisfy the budget and deadline constraints, respectively. DBCS is used to obtain a feasible scheduling by considering the budget and deadline constraints of the application simultaneously. HBCS and the proposed algorithm (i.e., MSLBL) are used to minimize the schedule length while satisfying the budget constraint of the application.

This experiment is conducted to compare the AR of Fast Fourier transform and Gaussian elimination parallel applications for varying number of tasks. We limit the size of applications changed from 96 (small scale) to 2560 (large scale) when ρ is changed from 16 to 256 in the Fast Fourier transform applications, and changed from 77 (small scale) to 2627 (large scale) when ρ is changed from 12 to 72 in the Gaussian elimination parallel applications. The budget constraint of applications $cost_{\text{bud}}(G)$ is considered in the range of $cost_{\min}(G) \times 1.1$ to $cost_{\min}(G) \times 10$. The deadline constraint of applications is defined in the range of $SL_{\text{HEFT}}(G) \times 1.1$ to $SL_{\text{HEFT}}(G) \times 10$, where $SL_{\text{HEFT}}(G)$ is the schedule length of an application using the HEFT algorithm. The total number of experiments is 1000.

Table 11 shows that the AR using the three algorithms decreases with the increase of the number of tasks in the Fast Fourier transform applications. However, the AR using MSLBL is significantly higher than those using HBCS and DBCS in the same scale. For example, when $|N| = 1152$, the AR using MSLBL and HBCS are 37% and 10%, respectively. An interesting phenomenon is that the ARs obtained by using HBCS and DBCS are almost the same. This is mainly because these two algorithms adopt the preassignment scheme with the minimum execution cost of tasks.

Compared with the results in Table 11, the change trend of the AR of Gaussian elimination applications in Table 12 is similar to that of Fast Fourier transform applications. However, the ARs using the three algorithms change slowly with the increase of the number of tasks. The AR using MSLBL is higher than those using HBCS and DBCS, and the difference between them increases with the increase of the number of tasks. The ARs using HBCS and DBCS are roughly the same in the same scale levels. The results indicate that MSLBL obtains the minimum schedule length under budget and deadline constraints and has a better acceptance rate than the other two algorithms.

The combination of all the results of Fast Fourier transform and Gaussian elimination parallel applications with budget and budget-deadline constraints indicates that the proposed MSLBL algorithm is very effective in schedule length minimization while satisfying the budget constraints.

6. Conclusions

An algorithm of schedule length minimization MSLBL for budget constrained parallel applications on heterogeneous cloud computing systems is developed in this study. First, the proposed algorithm can always satisfy the budget constraint, and the correctness is verified using proof and experiments. Second, the proposed MSLBL algorithm implements efficient and low time complexity task scheduling to minimize the schedule length. The MSLBL algorithm using the budget-level design technique is highly efficient in satisfying the budget constraint and minimizing the schedule length of different scales of real parallel applications compared with the existing budget and budget-deadline constrained algorithm. The proposed MSLBL algorithm could effectively improve part of the cost-aware design for parallel applications on heterogeneous cloud computing systems.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China with Grant No. 61672217, 61602164, the China Postdoctoral Science Foundation under Grant No. 2016M592422.

References

[1] M.W. Convolbo, J. Chou, Cost-aware DAG scheduling algorithms for minimizing execution cost on cloud resources, J. Supercomput. 72 (3) (2016) 985–1012.
[2] Y. Kong, M. Zhang, D. Ye, A belief propagation-based method for task allocation in open and dynamic cloud environments, Knowl.-Based Syst. 115 (2017) 123–132.
[3] J. Broberg, S. Venugopal, R. Buyya, Market-oriented grids and utility computing: The state-of-the-art and future directions, J. Grid Comput. 6 (3) (2008) 255–276.
[4] C.Q. Wu, X. Lin, D. Yu, et al., End-to-end delay minimization for scientific workflows in clouds under budget constraint, IEEE Trans. Cloud Comput. 3 (2) (2015) 169–181.
[5] Z. Wu, X. Liu, Z. Ni, et al., A market-oriented hierarchical scheduling strategy in cloud workflow systems, J. Supercomput. 63 (1) (2013) 256–293.
[6] H. Topcuoglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[7] X. Zhou, Y. Wang, X. Huang, C. Peng, Fast on-line task placement and scheduling on reconfigurable devices., in: FPL, 2007, pp. 132–138.
[8] M.A. Rodriguez, R. Buyya, Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds, IEEE Trans. Cloud Comput. 2 (2) (2014) 222–235.
[9] S. Abrishami, M. Naghibzadeh, D.H.J. Epema, Cost-driven scheduling of grid workflows using partial critical paths, IEEE Trans. Parallel Distrib. Syst. 23 (8) (2012) 1400–1414.
[10] F. Wu, Q. Wu, Y. Tan, et al., PCP-B^2: Partial critical path budget balanced scheduling algorithms for scientific workflow applications, Future Gener. Comput. Syst. 60 (2016) 22–34.

[11] H. Arabnejad, J.G. Barbosa, A budget constrained scheduling algorithm for workflow applications, J. Grid Comput. 12 (4) (2014) 665–679.
[12] H. Arabnejad, J.G. Barbosa, R. Prodan, Low-time complexity budget-deadline constrained workflow scheduling on heterogeneous resources, Future Gener. Comput. Syst. 55 (2016) 29–40.
[13] M. Malawski, G. Juve, E. Deelman, et al., Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaasclouds, Future Gener. Comput. Syst. 48 (2015) 1–18.
[14] T.C. Hu, Parallel sequencing and assembly line problems, Oper. Res. 9 (6) (1961) 841–848.
[15] G. Xie, G. Zeng, L. Liu, R. Li, K. Li, High performance real-time scheduling of multiple mixed-criticality functions in heterogeneous distributed embedded systems, J. Syst. Archit. (2016) 1–13.
[16] S. Abrishami, M. Naghibzadeh, D.H.J. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Gener. Comput. Syst. 29 (1) (2013) 158–169.
[17] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, X. Huang, Enhanced energy-efficient scheduling for parallel applications in cloud, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (ccgrid 2012), IEEE Computer Society, 2012, pp. 781–786.
[18] X. Xiao, G. Xie, R. Li, K. Li, Minimizing schedule length of energy consumption constrained parallel applications on heterogeneous distributed systems. in: 14th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2016, 2016, pp. 1471–1476.
[19] L. Yang, J. Cao, S. Tang, N. Suri, Run time application repartitioning in dynamic mobile cloud environments, IEEE Trans. Cloud Comput. 4 (3) (2016) 336–348.
[20] I.S. Rachel, J.S. Raj, V. Vasudevan, A reliable schedule with budget constraints in grid computing, Int. J. Comput. Appl. 64 (3) (2013) 23–35.
[21] Y. Ju, R. Buyya, C.K. Tham, QoS-based scheduling of workflow applications on service grids, in: Proc. of 1st IEEE International Conference on e-Science and Grid Computing, 2005.
[22] G. Xie, R. Li, K. Li, Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems, J. Parallel Distrib. Comput. 83 (2015) 1–12.
[23] A.C. Zhou, B. He, C. Liu, Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds, IEEE Trans. Cloud Comput. 4 (1) (2016) 34–48.
[24] Y. Yuan, X. Li, Q. Wang, et al., Deadline division-based heuristic for cost optimization in workflow scheduling, Inform. Sci. 179 (15) (2009) 2562–2575.
[25] W. Zheng, R. Sakellariou, Budget-deadline constrained workflow planning for admission control, J. Grid Comput. 11 (4) (2013) 633–651.
[26] W. Wang, Q. Wu, Y. Tan, et al., Maximize throughput scheduling and cost-Fairness optimization for multiple DAGs with deadline constraint, in: Algorithms and Architectures for Parallel Processing, Springer International Publishing, 2015, pp. 621–634.
[27] Q. Liu, W. Cai, J. Shen, et al., A speculative approach to spatial–temporal efficiency with multi-objective optimization in a heterogeneous cloud environment, Secur. Commun. Netw. 9 (17) (2016) 4002–4012.
[28] F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: a survey, J. Supercomput. (2015) 1–46.
[29] E.N. Alkhanak, S.P. Lee, R. Rezaei, et al., Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues, J. Syst. Softw. 113 (2016) 1–26.
[30] S. Singh, I.A. Chana, survey on resource scheduling in cloud computing: Issues and challenges, J. Grid Comput. 14 (2) (2016) 217–264.
[31] Z. Fu, X. Sun, Q. Liu, et al., Achieving efficient cloud search services: multi-keyword ranked search over encrypted cloud data supporting parallel computing, IEICE Trans. Commun. 98 (1) (2015) 190–200.
[32] S. Bharathi, A. Chervenak, E. Deelman, et al. Characterization of scientific workflows, in: The 3rd Workshop on Workflows in Support of Large Scale Science, 2008, pp. 1–10.

Weihong Chen is an Associate Professor in Hunan City University, China. She is current toward her Ph.D. degree in computer science and engineering in Hunan University. Her main research interests include intelligent computing, cloud computing, and parallel and reliable systems.

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.

Yang Bai received her B.S. and M.S. degrees from Hunan University in 2013 and 2016, respectively. She is currently working on the Ph.D. degree at Hunan Province, Hunan University. Her research interests include automotive embedded systems, automotive cyber–physical systems.

Chunnian Fan received her Ph.D. degree in computer science from Nanjing University in 2011. Her research interests include parallel and distributed system, pattern recognition and image processing.

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 430 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, Journal of Parallel and Distributed Computing. He is an IEEE Fellow.