Scheduling Real-Time Parallel Applications in Cloud to Minimize Energy Consumption

Biao Hu, *Member, IEEE*, Zhengcai Cao, *Member, IEEE*, and Mengchu Zhou, *Fellow, IEEE*

Abstract—Cloud computing has become an important paradigm in which scalable resources such as CPU, memory, disk and IO devices can be provided to users to remotely process their applications. In a cloud computing platform, energy consumption accounts for a significant cost portion. This article thus aims to present an energy-efficient scheduling algorithm for processing a user application with a real-time requirement. This problem is formulated as a non-linear mixed integer programming problem. We start with providing an optimal closed-form solution to its relaxation problem that aims to minimize the energy consumption without considering real-time requirements. To meet real-time requirements, we propose how to adjust task placement and resource allocation by making a good tradeoff between energy consumption and task execution time. Lastly, we find two equivalent optimal resource allocation strategies once task placement has been done. We then propose to adjust the start time of task execution such that an application's completion time can be further shortened. Experimental results on two real-case enchmarks and extensive synthetic applications demonstrate that our proposed method finds a schedule that generally has 30 and 20 percent less energy consumption than enhancement heterogeneous earliest finish time (E-HEFT) and genetic algorithm, respectively. Besides, the proposed method has a higher rate to successfully find a feasible schedule than them, and its computation time is close to E-HEFT's, but far less than the genetic algorithm's.

Index Terms—Energy consumption minimization, cloud computing, optimization methods, parallel application, real-time scheduling

1 Introduction

Cloud computing is an infrastructure-based and service-oriented computing model that provides users with on-demand computing capabilities [1], [2], [3]. With modern virtualization techniques that support flexible job allocation and dynamic resource allocation, cloud service users only need to submit their requests to a cloud computing system. Based on user requests, a cloud system automatically provides necessary hardware resources to process them; in this way users can break away from setting up and maintaining a computing platform, and focus more on their computing applications [4]. In this scenario, the central problem for a cloud service provider is how to properly allocate hardware infrastructures and schedule applications, such that, their processing cost can be minimized.

A significant part of cost in a large-scale cloud computing system is owing to energy consumption at various levels of computational processes. As an example, the Tianhe-2 supercomputer with 16,000 computing nodes is the world's fastest computer in 2014 and 2015, and consumes 17,808 kw of power [5]. The high energy consumption not only increases cost, but also has a bad effect on system performance such as reducing its reliability, and has a bad environmental effect such as increasing carbon emission. Currently, how to minimize energy consumption has become a major concern for many cloud computing providers to design and optimize their systems [6], [7], [8], [9], [10].

On the other side, minimizing the user tasks' completion time is also important. A short completion time means that users can get the computed result in a short time, thus saving their work hours. From a provider's perspective, this means higher system throughput and thus higher profit. However, it is often a conflicting problem to improve cloud real-time processing capability while minimizing its cost in energy consumption, because reducing task completion time often entails the use of more hardware infrastructures, thus incurring more energy consumption. To resolve this conflict, a scheduling algorithm is demanded to efficiently allocate and utilize cloud resources.

In this paper, we tackle this challenging problem, and focus on scheduling a user application composed of parallel and dependent tasks with data synchronization. Such application is of a general computation type that can be modeled as a Directed Acyclic Graph (DAG). The nodes in a DAG represent individual tasks and directed edges represent inter-task data dependencies between a pair of tasks [11]. A large number of user applications can be categorized to DAG models, such as fourier transformation [12], gaussian elimination [13] and image processing applications [14], [15]. A cloud computing platform is composed of servers, and each server can be further discomposed into virtual machines. The execution model in cloud is presented in Fig. 1, where each task in an application should be placed on a physical server and processed in a virtual machine created for it. With the

- *B. Hu and Z. Cao are with the College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China. E-mail: hubiao@mail.buct.edu.cn, giftczc@163.com.*
- *M. Zhou is with the Helen and John C. Hartmann Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA and also with Center of Research Excellence in Renewable Energy and Power Systems, King Abdulaziz University, Jeddah 21589, Saudi Arabia. E-mail: zhou@njit.edu.*

Manuscript received 26 May 2019; revised 20 Oct. 2019; accepted 21 Nov. 2019. Date of publication 28 Nov. 2019; date of current version 8 Mar. 2022. (Corresponding authors: Zhengcai Cao and MengChu Zhou.)
Recommended for acceptance by A. Zomaya.
Digital Object Identifier no. 10.1109/TCC.2019.2956498

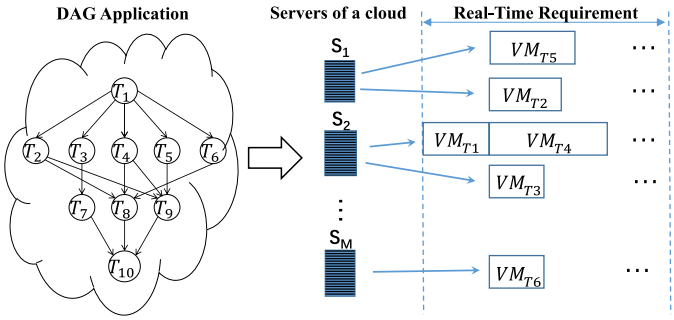


Fig. 1. Cloud computing execution model, where each task of a DAG application is processed in a virtual machine created in a server [17].

aim of minimizing energy consumption while meeting the real-time computing requirement, we have to answer the following three questions:

1) How to place tasks in a DAG application to servers?
2) When to execute each task to meet the task dependency requirement?
3) How to allocate resources to the virtual machine created for each task?

Answering the above questions is not easy. It has been well-known that scheduling tasks with the aim of minimizing task completion time is an NP-hard problem [16]. So is our concerned problem. Engineers have to reconcile the conflicting requirements of reducing energy consumption and shortening response time, because their interplay is tight and complicated. Intuitively, smaller energy consumption could increase the response time of a parallel workflow, and the problem of minimizing energy consumption and response time is considered as a typical bicriteria optimization problem.

Currently, there are already some scheduling approaches that partly answer the questions raised above. In [17], a scheduling algorithm is proposed to minimize job completion time with energy consumption constraints. Instead of scheduling a DAG application, the scheduled job in [17] is composed of a set of independently parallel tasks without data synchronization. In [18], an energy and deadline-aware task scheduling method is proposed to schedule data-intensive applications. This method mostly focuses on task assignments on virtual machines. Given a certain requirement of quality of service (QoS), a virtual machine scheduling algorithm is proposed in [19] to regulate allocated computing resources to reach the optimal energy level. Differing from our problem in which virtual machines are not decided beforehand, it assumes that virtual machines have been pre-mapped to servers. While several other scheduling approaches have also been presented to schedule applications in a cloud computing system, such as [20], [21], [22], none of them studies the task scheduling for a DAG application that needs to decide task placement, the start time of executing a task and the resource allocation in a virtual machine together. Besides, many previous approaches such as a genetic algorithm and particle swarm optimization [23], [24], [25], [26], [27] find a schedule solution through time-consuming iterative search, which could be impracticable in a cloud environment for some large-size parallel applications that need to be fast processed.

Being aware of the above problem, in this paper, based on a closed-form solution for the relaxation problem that aims to minimize energy consumption without timing requirements, we propose a scheduling algorithm that can quickly provide a solution for the above raised questions. Our main contributions are as follows:

- First, we formulate the scheduling problem as a non-linear mixed-integer programming problem. Based on the energy model, we find an optimal closed-form solution of placing tasks on servers to minimize its energy consumption.
- Second, to maintain data synchronization among tasks, we classify and schedule tasks at the application levels. Due to a real-time constraint, the task placement for minimizing energy consumption may make an application's completion time violate its timing constraint. Then, based on the tradeoff between completion time and energy consumption, we propose an algorithm that increases a relatively small energy to achieve a relatively great reduction of completion time.
- Third, once task placement has been done, we find two equivalent optimal resource allocation strategies to minimize an application's completion time, by noting that creating multiple virtual machines with optimal resource allocation to execute tasks in parallel is equivalent to providing resources to only one virtual machine to execute tasks sequentially. Thus, we propose an algorithm to adjust the start time of task execution in a backward way to further shorten an application's completion time.
- Last, we evaluate the performance of our proposed algorithms by conducting extensive simulations. The results demonstrate that, compared to a genetic algorithm, ours take much less computation to find a schedule with less energy consumption, and higher success rate.

Section 3 presents our problem formulation. Section 4 gives a task placement approach able to minimize energy consumption by ignoring timing requirements. Section 5 shows how to make adjustments towards task placement and start time of their execution to meet timing requirements. Experimental results are presented in Section 6. Section 7 concludes this paper.

2 Related Work

There have been many studies that investigate the task scheduling problem in clouds. We hereby survey some of them that we think highly relevant to our studied problem. The scheduling algorithms can often be categorized into two types: intelligent optimization algorithms such as genetic algorithm and ant colony optimization, and heuristic algorithms.

An intelligent optimization algorithm can often find a good solution because it relies on an iterative search. In [28], a genetic algorithm with two fitness functions is proposed to adaptively adjust the energy and performance target before task assignments in a cloud, in which way users can meet their own requirements. Zuo *et al.* [29] propose an improved ant colony algorithm to solve a bicriteria optimization problem to minimize makespan and a user's expense.

An algorithm integrating a genetic algorithm and simulated annealing is proposed in [30] where the offsprings from the former are optimized by using the latter. In [31], a clonal selection algorithm is adopted to minimize energy consumption and task processing time. Results from CloudSim show that it can achieve a good balance between the two indices. All these algorithms demand much computation to decide a schedule, which may not be applicable for cloud scenarios with fast timing response requirements.

Heuristics can often quickly provide a scheduling solution because they only use some simple rules to decide a schedule. In [18], by modeling datasets and tasks as a binary tree, a tree-to-tree task scheduling algorithm is proposed to improve the energy efficiency of a cloud system. In order to minimize energy consumption on a host, an optimal utilization level is introduced in [19] to execute a certain number of instructions. At this level, a virtual machine scheduling algorithm is proposed to regulate computing resource allocation. To balance different types of workloads and improve a system's overall utilization, a set of heuristics are proposed in [32] to execute tasks with high energy efficiency. Baker *et al.* [33] propose to use a bin-packing technique to select energy-efficient services from cross-continental clouds, in which way energy consumption can be reduced. A two-stage strategy is proposed in [34] that first makes use of historical scheduling data to classify arriving tasks and then creates virtual machines ahead, such that non-reasonable task allocation can be minimized in cloud. Due to the widespread deployment of IoT devices, the strategies of offloading requests from a center to edges become important. In [35], a fog computing framework is proposed to permit data processing by collaborating fog nodes. Hence, IoT requests can be quickly served and QoS is improved. However, these approaches either ignore real-time requirements of various applications, or make some presumptions of task placements and resource allocation, which is not able to solve our studied problem well.

3 Problem Formulation

To schedule an application submitted to a cloud computing system, each of its tasks should be placed on a server, and a virtual machine should be created to execute it. We formulate this scheduling problem with inputs, outputs, an objective function and related constraints.

3.1 Inputs

In a cloud computing system, inputs include a user application and a platform with a set of heterogeneous servers.

3.1.1 User Applications

An application composed of a set of parallel and dependent tasks can be represented as a directed acyclic graph model. Mathematically, it is $G(\mathcal{T}, E)$ where $\mathcal{T}$ is a set of tasks; E is a set of edges that represent task dependencies. Specifically, $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$ where T_i denotes the ith task. E is a $N \times N$ matrix where $E(i,j) = 1$ denotes the dependency constraint between T_i and T_j, i.e., T_j can start its execution only after T_i completes its execution. If there is no such dependency, $E(i,j) = 0$. We use L_i to represent the computing load of task T_i, i.e., it needs execution time of L_i on a virtual machine with unit computing capacity created from a unit-efficiency server.

TABLE 1
Main Notations in This Study

Notation	Definition
Constants	
$G(\mathcal{T}, E)$	the DAG application
CCP	the cloud computing platform
N	the task count in $G(\mathcal{T}, E)$
M	the server count in cloud computing platform
T_i	the ith task in DAG
L_i	the load of the task T_i
S_j	the jth server in cloud computing platform
C_j	the available computing resource on S_j
V_i	the virtual machine created for T_i
$\alpha_{i,j}$	power constant of processing T_i on server S_j
$\lambda_{i,j}$	the execution efficiency of task T_i on server S_j
$A_{j,k}$	the coefficient of kth resource on server S_j
$B_{j,k}$	the amount of kth resource on server S_j
E_i	the energy consumption of ith virtual machine
$\mathcal{R}$	the real-time requirement
$\mathcal{T}_l$	the set of lth level tasks
$\mathcal{T}_l^j$	tasks on jth server and at lth level
Variables	
$x_{i,j}$	the placement indicator of T_i on server S_j
c_i	the amount of computing resources to T_i
s_i	the start execution time of task T_i
e_i	the execution time of task T_i
$\mathcal{T}_l^j$	the set of lth level tasks placed on jth server
$\mathcal{S}$	the schedule output

In addition, we define two special tasks in DAG, which are an entry task and an exit task. The former is a task without any predecessor task, and the latter is one without any successor task. If there is more than one task who has no predecessor task, we create a dummy task to be their predecessor task. This scheme also applies to multiple exit tasks. In this way there must be only one entry task and only one exit one. Tasks are labeled starting from the entry task to the exit task, i.e., $T_{entry} = T_1$ and $T_{exit} = T_N$. Other important notations can be found on Table 1.

3.1.2 Cloud Computing Platform

A cloud computing platform can be treated as a set of heterogeneous servers that provide different computing capabilities. We denote them as $CCP = \{S_1, S_2, \ldots, S_M\}$ where S_j is the jth sever in this platform. Specifically, we use C_j to denote the computing resource of server S_j. It should be noted that, because of the server heterogeneity, the execution time of a task is different on different servers [36]. We introduce the execution efficiency of a task to denote such heterogeneity. The execution efficiency of task T_i on sever S_j is denoted as $\lambda_{i,j}$. Then, we have

$$e_i = \frac{L_i}{\lambda_{i,j} c_i}, \tag{1}$$

where L_i is the task execution load, and e_i is the execution time of task T_i on a virtual machine allocated with c_i computing capacity from server S_j.

3.2 Output

The outputs of scheduling an application in a cloud contains a task placement plan, start time plan and resource allocation plan.

3.2.1 Task Placement Plan

The server assigned to execute a specific task is denoted by a task placement plan. For task T_i, we can use binary variables $x_{i,j}$ to denote its placement, where $x_{i,j} = 1$ denotes that T_i is placed on server S_j to execute, and $x_{i,j} = 0$ otherwise. Here we should note that a task is non-divisible, which means that it can only be placed on one server at a time. The task placement plan can be represented by a matrix $\mathbf{X}$ with a size $N \times M$.

3.2.2 Start Time Plan

Due to the task dependency in an application, a task can start its execution only after all its predecessor tasks have been finished. Here, we use $\overrightarrow{s} = \{s_1, s_2, \ldots, s_N\}$ to denote their start time, where s_i denotes the time to start T_i's execution.

3.2.3 Resource Allocation Plan

A task is processed by a virtual machine, where a virtual machine is created from a server. A resource allocation plan denotes the number of computing resources allocated to each virtual machine. Here we use $\mathcal{V} = \{V_1, V_2, \ldots, V_N\}$ to denote the set of virtual machines, where V_i is the virtual machine created for T_i. Correspondingly, we use $\overrightarrow{c} = \{c_{t_1}, c_{t_2}, \ldots, c_{t_N}\}$ to denote a resource allocation plan in which c_i is the number of computing resources allocated to V_i. In summary, an output schedule can be represented by $\mathcal{S} = \{\mathbf{X}, \overrightarrow{s}, \overrightarrow{c}\}$.

3.3 Objective

Our goal is to propose some approaches that can quickly find good solutions to minimize energy consumption. Before we present energy consumption minimization as an objective function, we briefly present a power model used in cloud computing.

We adopt a power model that is widely used in cloud computing energy analysis [4], [28], [37], [38], [39], [40]. The power consumption of a virtual machine created for processing task T_i, P_i, is directly proportional to its resource utilization, i.e.,

$$P_i = \alpha_{i,j} \cdot c_i, \tag{2}$$

where $\alpha_{i,j}$ is a specific power constant that only depends on a virtual machine itself. Then, the energy consumption E_i of the ith virtual machine is

$$E_i = P_i \cdot e_i = \alpha_{i,j} \cdot c_i \cdot e_i = \frac{\alpha_{i,j} L_i}{\sum_{j=1}^{M} x_{i,j} \lambda_{i,j}}. \tag{3}$$

where $\sum_{j=1}^{M} x_{i,j}\lambda_{i,j}$ denotes the execution efficiency of task T_i on its allocated server.

Here, we use $\mathcal{E} = \{E_1, E_2, \ldots E_N\}$ to denote the set of energy consumption of all created virtual machines, and the scheduling objective is to minimize the total energy consumption that can be presented as below

$$\min \sum_{i=1}^{N} E_i. \tag{4}$$

3.4 Constraints

There are a variety types of constraints when scheduling an application in a cloud computing platform.

3.4.1 Task Placement Constraints

Because a task is non-divisible, a task can only be allocated to one sever at a time, i.e.,

$$x_{i,j} \in \{0, 1\} \text{ and } \sum_{j=1}^{M} x_{i,j} = 1,\ i = 1, .., N, j = 1, .., M. \tag{5}$$

3.4.2 Task Dependency Constraints

Due to the task dependency in an application, a task can start its execution only after its all predecessor tasks have completed their execution. Thus, we have

$$\begin{aligned} s_i &\leq \max_{T_{i'} \in pred(T_i)} \left(s_{i'} + e_{i'} \right) \\ &= \max_{T_{i'} \in pred(T_i)} \left(s_{i'} + \frac{L_i}{\sum_{j=1}^{M} x_{i',j} \lambda_{i',j} c_{i'}} \right), \end{aligned} \tag{6}$$

where $pred(T_i)$ represents the set of all predecessor tasks of task T_i, i.e., $pred(T_i) = \{T_{i'} | E(i', i) = 1, \ \forall T_{i'} \in \mathcal{T}\}$.

3.4.3 Real-Time Constraint

Suppose that the first task (entry task) of an application starts its execution time at 0, i.e., $s_1 = 0$. The application's completion time is the moment of completing its exit task. Due to the timing constraint, we have

$$s_N + e_N = s_N + \frac{L_N}{\sum_{j=1}^{M} x_{N,j} \lambda_{N,j} c_{t_N}} \leq \mathcal{R}, \tag{7}$$

where $\mathcal{R}$ represents its real-time requirement.

3.4.4 Resource Availability Constraints

In a cloud computing platform, there are specific number of computing resources available. We suppose that these resources have been specifically allocated to each sever. Then the allocated resources to virtual machines in a sever should not exceed its total quantity at any time. We find that a computing resource would be added to a newly created virtual machine only when a task needs to start its execution. Thus, the resource availability constraints only need to be checked at those moments, i.e.,

$$\begin{aligned} &A_{j,k} \cdot \sum_{T_i \in \mathcal{T}^j(t)} x_{i,j} \cdot c_i \leq B_{j,k}, \\ &k = 1, .., R;\ j = 1, .., M;\ t \in \{s_1, s_2, \ldots, s_N\}, \end{aligned} \tag{8}$$

where $A_{j,k}$ is the resource coefficient in the kth type constraint; $B_{j,k}$ is the total number of the kth type constraints in jth server; $\mathcal{T}^j(t)$ is the set of tasks that are placed and executed on the jth server at time t.

From the above problem description, we find that this scheduling problem is a complex non-linear mixed-integer programming problem, which is very hard to solve exactly. It also takes a long time by using some intelligent algorithms, such as genetic algorithm or differential evolution algorithm, to get a good solution. In this paper, we propose a novel heuristic method that is able to get an energy-efficient schedule in a short time, even for a large-scale scheduling problem.

4 Energy Consumption-Optimal Scheduling Without Timing Constraint

In this section, we relax our studied problem to be a problem of minimizing the energy consumption without a real-time requirement. This scheduling solution is next used as a reference for the problem with a real-time requirement.

4.1 Energy Consumption-Optimal Task Placement Plan

The energy consumption is the accumulated energy consumption of all tasks. Looking into (3) and (4), we can find out that the energy consumption of a task has the following property.

Lemma 1. *The energy consumption of task T_i ($T_i \in \mathcal{T}$) only depends on its placement.*

Proof. The energy consumption of task T_i is E_i presented in (3), where L_i is a parameter that is related to T_i itself. Then E_i only depends on $\frac{\alpha_{i,j}}{\sum_{j=1}^{M} x_{i,j}\lambda_{i,j}}$, which means that its energy consumption only depends on the server that T_i is placed on. □

Based on this lemma, we can further conclude the below theorem.

Theorem 1. *The energy consumption for executing an application is minimized if each of its tasks is placed on a server as follows:*

$$x_{i,j'} = 1,\ \forall i = 1\,..\,N, \tag{9}$$

where j' is subject to $\frac{\alpha_{i,j'}}{\lambda_{i,j'}} = \min_{j=1\,..\,M}\{\frac{\alpha_{i,j}}{\lambda_{i,j}}\}$.

Proof. For task T_i, its energy consumption depends on $\frac{\alpha_{i,j}}{\sum_{j=1}^{M} x_{i,j}\lambda_{i,j}}$ only. Because $x_{i,j}$ is a binary value and $\sum_{j=1}^{M} x_{i,j} = 1$, we have

$$\frac{\alpha_{i,j}}{\sum_{j=1}^{M} x_{i,j}\lambda_{i,j}} \geq \min_{j=1\,..\,M}\left\{\frac{\alpha_{i,j}}{\lambda_{i,j}}\right\} = \frac{\alpha_{i,j'}}{\lambda_{i,j'}}. \tag{10}$$

Thus, to minimize the value of $\frac{\alpha_{i,j}}{\sum_{j=1}^{M} x_{i,j}\lambda_{i,j}}$, $x_{i,j'}$ must be equal to 1. Because the energy consumption of each task is independent by Lemma 1, the total energy consumption is minimized based on (9). □

From this theorem, we can decide the energy consumption-optimal task placement plan based on $\alpha_{i,j}$ and $\beta_{i,j}$ ($i \leq N$, $j \leq M$).

4.2 Partition Scheduling

With a task placement plan, we need to determine a start time plan and resource allocation plan. Due to task dependencies, a task can start its execution only after all its predecessors have been completed. To sufficiently guarantee this property, a scheme is designed to partition a schedule based on an application's DAG structure.

Definition 1. *In DAG, we call a task at its lth level if the maximum number of edges from its entry task to it is $l-1$.*

In addition, we use $\mathcal{T}_l$ ($l \leq L$) to denote the set of tasks at the lth level and L the maximum level. With this definition, we know that tasks at the same level have no dependency. Thus they can be processed in parallel. If a schedule runs tasks in the order of $\mathcal{T}_1, \mathcal{T}_2, \ldots$, and $\mathcal{T}_L$, task dependencies are guaranteed because processors of a task only stay at its upper levels. Such scheme leads to a partition schedule.

Then, we analyze how to schedule tasks at the same level. In fact, there is an optimal resource allocation strategy to minimize their execution time. First, we define $\mathcal{T}_l^j$ representing tasks on the jth server and at the lth level. It can be decided based on Theorem 1.

Theorem 2. *To minimize the completion time of $\mathcal{T}_l^j$, the computing resource allocated to T_i ($T_i \in \mathcal{T}_l^j$) should be*

$$\forall T_i \in \mathcal{T}_l^j,\ c_i^* = \frac{L_i}{\lambda_{i,j}\sum_{T_p\in\mathcal{T}_l^j}\frac{L_p}{\lambda_{p,j}}} \cdot \min_{k\in\{1\ldots R\}}\left\{\frac{B_{j,k}}{A_{j,k}}\right\}. \tag{11}$$

Proof. As there is no dependency at the same level, tasks in $\mathcal{T}_l^j$ can start their execution simultaneously, and the completion time of $\mathcal{T}_l^j$ depends on the maximum execution time of its tasks.

First, we denote that $\frac{B_{j,u}}{A_{j,u}} = \min_{k=1..R}\left\{\frac{B_{j,k}}{A_{j,k}}\right\}$. We have

$$\begin{aligned} c_i^* &= \frac{L_i}{\lambda_{i,j}\sum_{T_p\in\mathcal{T}_l^j}\frac{L_p}{\lambda_{p,j}}} \cdot \min_{k=1..R}\left\{\frac{B_{j,k}}{A_{j,k}}\right\} \\ &= \frac{L_i}{\lambda_{i,j}\sum_{T_p\in\mathcal{T}_l^j}\frac{L_p}{\lambda_{p,j}}} \cdot \frac{B_{j,u}}{A_{j,u}}. \end{aligned}$$

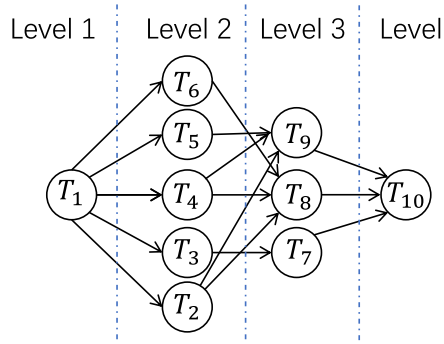
We then prove that with the resource allocation plan given in (11), all tasks in $\mathcal{T}_l^j$ have the same execution time, because for any two tasks T_i and $T_{i'}$ ($T_i \in \mathcal{T}_l^j, T_{i'} \in \mathcal{T}_l^j, i \neq i'$),

$$\begin{aligned} e_i^* &= \frac{L_i}{\lambda_{i,j}c_i^*} = \frac{L_i}{\lambda_{i,j}} \cdot \frac{\lambda_{i,j}}{L_i} \cdot \sum_{T_p\in\mathcal{T}_l^j}\frac{L_p}{\lambda_{p,j}} \cdot \frac{A_{j,u}}{B_{j,u}}, \\ &= \frac{L_{i'}}{\lambda_{i',j}} \cdot \frac{\lambda_{i',j}}{L_{i'}} \cdot \sum_{T_p\in\mathcal{T}_l^j}\frac{L_p}{\lambda_{p,j}} \cdot \frac{A_{j,u}}{B_{j,u}} = e_{i'}^*. \end{aligned}$$

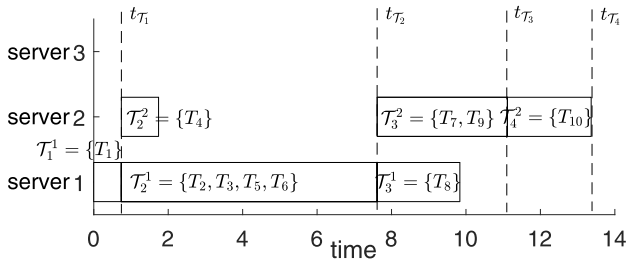
Last, we prove that c_i^* is the optimal solution for minimizing the completion time of $\mathcal{T}_l^j$. We prove it by contradiction. We know that the completion time of $\mathcal{T}_l^j$ with c_i^* is e_i^*. Now assume that instead of c_i^*, another resource allocation $c_i^\flat$ makes the completion time smaller, i.e.,

$$e_i^\flat = \frac{L_i}{\lambda_{i,j}c_i^\flat} \leq \frac{L_i}{\lambda_{i,j}c_i^*} = e_i^*,$$

then we have $c_i^\flat > c_i^*$. In this case, the total number of required resources ($\forall k \in [1..R]$) is



(a) A DAG with 10 tasks divided into 4 levels



(b) Scheduling results

Fig. 2. An illustrative example of scheduling a DAG with 10 tasks and 4 levels by Algorithm 1, where every task in $\mathcal{T}_l^j$ ($l \le L$, $j \le M$) has the same start and end time.

$$A_{j,k} \cdot \sum_{T_i \in \mathcal{T}_l^j} c_i^\flat > A_{j,k} \cdot \sum_{T_i \in \mathcal{T}_l^j} c_i^*$$
$$= A_{j,k} \cdot \sum_{T_i \in \mathcal{T}_l^j} \frac{L_i}{\lambda_{i,j} \sum_{T_p \in \mathcal{T}_l^j} \frac{L_p}{\lambda_{p,j}}} \cdot \min_{k=1..R} \left\{ \frac{B_{j,k}}{A_{j,k}} \right\}.$$

Considering the uth resource constraint, we have

$$A_{j,u} \cdot \sum_{T_i \in \mathcal{T}_l^j} c_i^\flat > A_{j,u} \cdot \frac{\sum_{T_i \in \mathcal{T}_l^j} \frac{L_i}{\lambda_{p,j}}}{\sum_{T_p \in \mathcal{T}_l^j} \frac{L_p}{\lambda_{p,j}}} \cdot \frac{B_{j,u}}{A_{j,u}} = B_{j,u},$$

which violates the resource availability constraint. As a result, there does not exist any resource allocation plan that is better than that in (11). □

Given a resource allocation plan, we can decide the execution time of each task. As the schedule is partitioned level by level, we have

$$s_i = t_{\mathcal{T}_{l-1}}$$
$$t_{\mathcal{T}_l} = t_{\mathcal{T}_{l-1}} + e_{\mathcal{T}_l},$$

where

$$e_{\mathcal{T}_l} = \max\{e_i | \forall T_i \in \mathcal{T}_l\},$$

and $t_{\mathcal{T}_l}$ denotes the moment of completing all tasks in $\mathcal{T}_l$ as $t_{\mathcal{T}_l}$.

The detailed scheduling procedure is presented in Algorithm 1. Here we use a specific example to explicitly show the scheduling results.

Example 1. For a DAG with 10 tasks divided into 4 levels (Fig. 2a), its parameters are presented in Table 2. Suppose the real-time requirement is 9. After applying Algorithm 1, we have the schedule shown in Fig. 2b, where the execution of any task in $\mathcal{T}_l^j$ ($l \le L$, $j \le M$) starts and ends simultaneously due to their equal execution time. The application's completion time is 13.36 and the energy consumption is 2,755. This schedule needs to be further adjusted due to the timing requirement.

TABLE 2
Parameters of the DAG and Cloud Computing Platform in the Example

L_i

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
200	300	400	200	300	500	200	400	500	500

$\lambda_{i,j}$

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
S_1	0.9	0.9	0.5	0.8	0.7	1.0	0.5	0.6	0.6	0.5
S_2	0.8	0.6	0.6	0.9	0.6	0.7	0.9	0.6	0.9	1.0
S_3	0.7	0.5	0.5	0.5	0.9	0.8	0.7	0.9	0.6	1.0

$\alpha_{i,j}$

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
S_1	0.7	0.5	0.6	1.0	0.5	1.0	0.5	0.5	0.7	0.8
S_2	0.7	1.0	0.9	1.0	0.5	0.7	0.7	0.7	0.5	0.5
S_3	0.6	0.7	0.6	1.0	0.7	0.8	0.8	1.0	1.0	0.7

$A_{j,k}$

k	1	2	3	4	5	6	7
S_1	200	300	300	300	500	300	300
S_2	200	300	300	500	500	300	500
S_3	400	300	200	400	300	200	200

$B_{j,k}$

k	1	2	3	4	5	6	7
S_1	0.2	0.8	0.8	1.0	0.3	0.7	0.2
S_2	0.9	0.8	0.1	0.5	0.9	0.6	1.0
S_3	0.4	0.7	0.4	0.9	0.5	0.3	1.0

Algorithm 1. Energy Consumption-Optimal Schedule without Real-Time Constraint

1: **function** EnergyOptiSchedule($G(\mathcal{T}, E)$, CCP)
2: Decide $x_{i,j}$ ($\forall i = 1..N$) according to Theorem 1.
3: Based on the task placement plan and DAG structure, classify tasks to $\mathcal{T}_l^j$, where $j \le M$, $l \le L$.
4: **for** $l = 1..L$ **do**
5: **for** $j = 1..M$ **do**
6: $\forall T_i \in \mathcal{T}_l^j$, $s_i = t_{\mathcal{T}_{l-1}}$ and allocate the computing resource to T_i according to Theorem 2
7: **end for**
8: $t_{\mathcal{T}_l} = t_{\mathcal{T}_{l-1}} + \max\{e_i | \forall T_i \in \mathcal{T}_l\}$
9: **end for**
10: **Return** $\mathcal{S}$
11: **end function**

5 Scheduling With Real-Time Constraint

Although the energy consumption is minimized by Algorithm 1, its schedule may not meet real-time requirement. We have to find another approach that takes the timing requirement into account. Because the schedule from Algorithm 1 has the minimum energy consumption but not the minimum completion time, our idea is to make some adjustments to this schedule such that its completion time can be

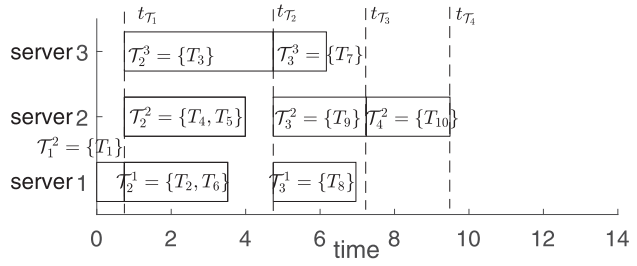


Fig. 3. Scheduling results by Algorithm 2.

scaled down with minimal extra energy consumption. In this section, based on the schedule from Algorithm 1, we present an efficient strategy to reduce the completion time in exchange of energy saving.

5.1 Task Placement Adjustment Strategy

Before making adjustments to task placement, we have to find out a potential task whose placement change can bring the largest benefit to its completion time reduction. Looking at the schedule from Algorithm 1 (such as Fig. 2b), task execution time is different in different servers. Thus, tasks with the maximum execution time at their levels should be placed onto other servers. Only in this way can an application's completion time be reduced. Such tasks that have the maximum execution time at each level are called *bottleneck tasks*. As an example, $\mathcal{T}_2^1$ and $\mathcal{T}_3^2$ in Fig. 2b are bottleneck tasks at levels 2 and 3.

Naturally, the best strategy is to reduce the completion time maximally at the least cost of energy consumption. Thus we have to find an appropriate indicator to characterise task. Before we provide it, we formally define the following notations. Suppose that a task T_i at the lth level has been placed on the jth server, i.e., $T_i \in \mathcal{T}_l^j$. Assume that this task is also a bottleneck task. If it is placed on the j'th server ($j \neq j'$), the increment of energy consumption is denoted as $\Delta E_i^{j \to j'}$ and the decrement of $e_{\mathcal{T}_l}$ is denoted as $\Delta e_{\mathcal{T}_l}^{j \to j'}$.

Definition 2. *The energy-to-time factor for task T_i is defined as Γ_{t_i} that is calculated by*

$$\Gamma_{t_i} = \begin{cases} \min_{j' \neq j,\, j' \leq M} \left\{ \dfrac{\Delta E_i^{j \to j'}}{\left| \Delta e_{\mathcal{T}_l}^{j \to j'} \right|} \right\}, & \text{if } \Delta e_{\mathcal{T}_l}^{j \to j'} < 0 \\ +\infty, & \text{otherwise.} \end{cases} \tag{12}$$

where

$$\Delta E_i^{j \to j'} = E_i^{j'} - E_i^j,$$
$$\Delta e_{\mathcal{T}_l}^{j \to j'} = e_{\mathcal{T}_l}^{j'} - e_{\mathcal{T}_l}^j,$$

where $E_i^{j'}$ and $e_{\mathcal{T}_l}^{j'}$ denote the energy consumption and execution time of T_i when T_i is placed on server j'.

We define Γ_{t_i} as the positive infinity when the adjustment does not shorten the execution time. In most cases, the smaller Γ_{t_i}, the greater ability T_i has to reduce completion time, which means that Γ_{t_i} reflects T_i's potential to reduce completion time. To better make task placement adjustment, we need to introduce the concept of a key bottleneck task at each level.

Definition 3. *The key bottleneck task T_{kb}^l at level l is the one whose energy-to-time factor is the smallest at its level, i.e.,*

$$\Gamma_{kb}^l = \min\{\Gamma_{t_i} | \forall T_i \in \mathcal{T}_l\}. \tag{13}$$

Besides, we use the tuple $\{T_{kb}^l, y_{kb}^l, y_{kb}^{l*}, \Gamma_{kb}^l\}$ to represent the placement adjustment that moves the key bottleneck task T_{kb}^l from server y_{kb}^l to server y_{kb}^{l*} to get Γ_{kb}^l. After calculating this tuple at each level, we can get a matrix as follows:

$$\Psi = \begin{bmatrix} T_{kb}^1 & y_{kb}^1 & y_{kb}^{1*} & \Gamma_{kb}^1 \\ T_{kb}^2 & y_{kb}^2 & y_{kb}^{2*} & \Gamma_{kb}^2 \\ \cdots & \cdots & \cdots & \cdots \\ T_{kb}^L & y_{kb}^L & y_{kb}^{L*} & \Gamma_{kb}^L \end{bmatrix}.$$

Task placement is adjusted based on Ψ. As presented in Algorithm 2, at the beginning, Ψ is calculated based on the schedule from Algorithm 1. Then a task with a minimum energy-to-time factor is placed from its original server to a new server according to Ψ. Meanwhile, Ψ and the schedule is updated. If the new schedule makes the application's completion time meet the required timing constraints, this algorithm successfully terminates. Otherwise, this procedure continues until there does not exist a bottleneck task that can shorten the execution time.

Algorithm 2. Task Placement Adjustment Taking Real-Time Constraint Into Account

function PLACEMENTADJUSTMENTFUNCTION($\mathcal{S}$, $G(\mathcal{T}, E)$, CCP)
  With schedule from Algorithm 1, calculate Ψ by (12), (13)
  while true **do**
    From Ψ, pick up the key bottleneck task T_{kb}^p with the minimum energy-to-time factor, i.e., $\Gamma_{t_{kb}^p} = \min\{\Gamma_{t_{kb}^i} | i = 1..L\}$
    Place the task T_{kb}^p from $y_{t_{kb}^p}$ th server to $y_{t_{kb}^p}^*$ th server and re-allocate computing resource by (11)
    Update the schedule and pth row of Ψ
    if $t_{\mathcal{T}_L} \leq \mathcal{R}$ **then**
      Return $\mathcal{S}$
    else if $\min\{\Gamma_{t_{kb}^i} | i = 1 .. L\} = +\infty$ **then**
      Return Failure
    end if
  end while
end function

For the example shown in Fig. 2, after applying Algorithm 1, we can get a schedule as shown in Fig. 3, where the application's completion time has been shortened to 9.49, and energy consumption has been increased to 2,864. Compared to the schedule of Fig. 2b, three tasks have been migrated to new servers, i.e., T_3 from server 1 to server 3, T_5 from sever 1 to server 2, and T_7 from server 2 to server 3. Because this schedule does not meet the timing requirement, it has to be further adjusted.

5.2 Start Time Adjustment

In our previous schedules, in order to sufficiently meet task dependency constraints, task execution is partitioned according to DAG levels. Although this guarantees task

dependencies, it is not a tight schedule because some tasks are unnecessarily delayed to execute. For example, T_9 can actually be started right after the completion of $\mathcal{T}_2^2$, which makes the application's completion time shorter than that of Fig. 3. In this section, we discuss how to tightly schedule all tasks after relaxing the assumption that tasks at the same level must be grouped together to start simultaneously.

We have proved that the energy consumption only depends on a task placement plan in Lemma 1. The adjustment on task start time pays no energy cost to shorten the completion time. Thus, following the schedule from Algorithm 2, we focus on adjusting task start time and computing resource allocation by keeping a task placement plan unchanged. We have derived an optimal solution of computing resource allocation when tasks are grouped together to start their execution simultaneously. In fact, when tasks are ungrouped to start their execution sequentially, there is also an optimal solution for computing resource allocation. It can even be proved that the two optimal solutions achieve same completion time.

Theorem 3. *Suppose that tasks in $\mathcal{T}_l^j$ start their execution sequentially. If the computing resource allocated to T_i ($T_i \in \mathcal{T}_l^j$) is*

$$c_i^\sharp = \min_{k=1..R}\left\{\frac{B_{j,k}}{A_{j,k}}\right\}, \tag{14}$$

the completion time equals that of allocating computing resource by (11) when tasks are simultaneously started.

Proof. The execution time of task T_i with $c_i^\sharp$ is

$$e_i^\sharp = \frac{L_i}{\lambda_{i,j}c_i^\sharp} = \frac{L_i}{\lambda_{i,j}\min_{k=1..R}\left\{\frac{B_{j,k}}{A_{j,k}}\right\}}.$$

Because tasks in $\mathcal{T}_l^j$ start their execution sequentially, the completion time of $\mathcal{T}_l^j$ is

$$\begin{aligned} e_{\mathcal{T}_l^j} &= \sum_{T_i\in\mathcal{T}_l^j} e_i^\sharp = \sum_{T_i\in\mathcal{T}_l^j}\frac{L_i}{\lambda_{i,j}c_i^\sharp} = \sum_{T_i\in\mathcal{T}_l^j}\frac{L_i}{\lambda_{i,j}\min_{k=1..R}\left\{\frac{B_{j,k}}{A_{j,k}}\right\}}, \\ &= \frac{L_i}{\lambda_{i,j}}\cdot\frac{\lambda_{i,j}}{L_i}\cdot\sum_{T_p\in\mathcal{T}_l^j}\frac{L_p}{\lambda_{p,j}}\cdot\frac{1}{\min_{k=1..R}\left\{\frac{B_{j,k}}{A_{j,k}}\right\}} \\ &= \frac{L_i}{\lambda_{i,j}c_i^*} = e_i^*. \end{aligned}$$

Therefore, we have proved that the resource allocation via (14) is equivalent to that via (11) in minimizing the completion time of $\mathcal{T}_l^j$. □

With the resource allocation plan for tasks to start individually, we need to determine a task execution order. Based on the schedule from Algorithm 2, such order can be determined successively from the first level to the last one. Our idea is to start task execution as soon as possible. Therefore, we need to decide at which time a task can start its execution. The available start time for each task can be determined by considering the execution of its predecessor tasks and the available execution time on its allocated server. Then, task execution order is ranked in the ascending order of the calculated available start time, i.e., the earlier a task's execution can start, the higher priority it gets executed ahead of others. If the available start time of some tasks are the same, a random order is generated among them.

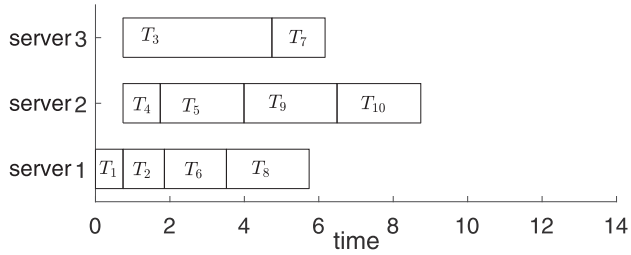


Fig. 4. Scheduling results by Algorithm 3.

The concrete procedure of task start time adjustment is presented in Algorithm 3. At the beginning, the available time of all servers is initialized to be 0. Then successively from the first level to the last one, task execution order depends on their predecessor task execution. Correspondingly, task start time is adjusted based on the new execution order. After applying this algorithm to the schedule in Fig. 3, a new schedule is obtained shown in Fig. 4, where every task starts its execution individually, and the application's completion time has been furthered shortened to 8.74 without increasing any energy consumption.

Algorithm 3. Task Start Time Adjustment Algorithm

```
function TaskStartTimeAdjustment(S, G(T, E), CCP)
  Initialize server available time as [avail_{S_1}, avail_{S_2}, ..., avail_{S_M}] = [0, 0, .., 0].
  for l = 1..L do
    for j = 1..M do
      for T_i ∈ T_l^j do
        Allocate the computing resource to T_i according to (14)
        est_i = max(avail_{S_j}, max_{T_{i'}∈pred(T_i)}(s_{i'} + e_{i'}))
      end for
      Rank the execution order of tasks in T_l^j in ascending order of est_i
      ∀T_i ∈ T_l^j, s_i = max(est_i, max_{T_{i'}∈hiPri(T_i)} (s_{i'} + e_{i'}))
      Suppose T_{i*} = highestPriority(T_l^j), avail_{S_j} = s_{i*} + e_{i*}
    end for
  end for
  Return S
end function
```

5.3 Schedule Adjustment in a Backward Way

We have presented task placement adjustment and start time adjustment to shorten an application's completion time, where the former needs to pay extra energy cost while the latter does not. From this viewpoint, it is better to apply the latter. However, the latter suffers the problem that it interrupts level-based partition scheduling scheme, thus making our task placement adjustment hard to apply.

To avoid this problem, our idea is to first aggressively adjust task placement using Algorithm 2, until a real-time constraint is met or there does not exist a bottleneck task to further shorten the completion time. In the case that the real-time constraint is met, tasks are returned to their previous servers one by one in the reverse order that tasks are placed in Algorithm 2. Here we denote the task placement

adjustment order in Algorithm 2 as $\{T_1^p, T_2^p, \ldots T_Z^p\}$ where T_1^p's placement is first adjusted and T_Z^p the last. Then, tasks are returned to the servers in the order of $\{T_Z^p, \ldots T_2^p, T_1^p\}$. Every time a task is returned we test whether the task start time adjustment of Algorithm 3 makes the completion time smaller than the required one or not. In this way the energy cost is reduced because tasks in their pervious servers consume less energy. If there does not exist a bottleneck task to further shorten the completion time after Algorithm 2, the algorithm of task start time adjustment is directly used to shorten the completion time.

The detailed procedure is given in Algorithm 4. At the beginning, we have a schedule from Algorithm 2. If it cannot meet real-time constraint, the task start time is directly adjusted. After this adjustment, the algorithm returns a feasible schedule or a failure report, as shown in Lines 15-20 of Algorithm 4. If the schedule from Algorithm 2 has already met the real-time constraint, tasks are progressively returned to their previous servers. Every time a task is returned, the schedule is tested whether the task start time adjustment is able to meet the timing requirement. If so, tasks are returned continuously. Otherwise, the task is stopped from returning, as shown in Lines 2-13 of Algorithm 4. The worst case that could happen to our task-return scheme is that no task can be returned without violating timing requirement. If no task is returned, the schedule is the same as the schedule from Algorithm 2. Because the timing requirement has already been met by the schedule, we can always find a feasible schedule that meets the timing requirement by executing code in Lines 2-13.

Algorithm 4. Schedule Adjustment in a Backward Way

1: **function** ScheduleAdjustmentBackward($\mathcal{S}$, G, CCP)
2: **if** $t_{T_L} \leq \mathcal{R}$ **then**
3: **for** $i = Z..1$ **do**
4: Return task T_i^p to its previous server.
5: $\mathcal{S} \leftarrow TaskStartTimeAdjustment(\mathcal{S}, G, CCP)$
6: **if** $t_{T_L} \leq \mathcal{R}$ **then**
7: Abandon the adjustment in Line 5.
8: **else**
9: Abandon returning task T_i^p in Line 4 and abandon the adjustment in Line 5.
10: $\mathcal{S} \leftarrow TaskStartTimeAdjustment(\mathcal{S}, G, CCP)$
11: **Return** $\mathcal{S}$
12: **end if**
13: **end for**
14: **else**
15: $\mathcal{S} \leftarrow TaskStartTimeAdjustment(\mathcal{S}, G, CCP)$
16: **if** $t_{T_L} \leq \mathcal{R}$ **then**
17: **Return** $\mathcal{S}$
18: **else**
19: **Return Failure**
20: **end if**
21: **end if**
22: **end function**

6 Performance Evaluation

Our proposed algorithm is evaluated through simulations, and in the following, we present our simulation setup, compared approaches and simulation results.

6.1 Simulation Setup

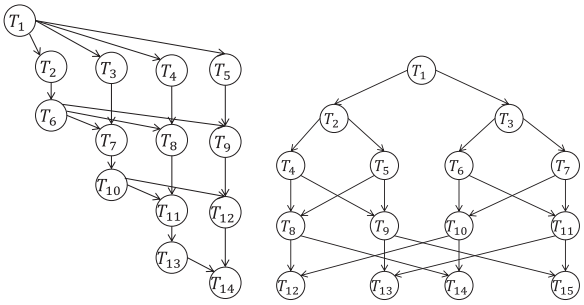
The presented approaches in this paper aim to provide an energy-efficient schedule in a cloud computing platform with a concrete application. Thus, in simulations, we should generate an application and a cloud computing platform with a specific number of hardware resources and other performance parameters. Here we use Gaussian elimination and fast Fourier transform as two real-case applications in the evaluation. In addition, we also generate random applications for more general evaluation.

We suppose that a cloud computing platform is a data center with M servers and R resource types. A task load L_i in an application is a number uniformly generated from (0, 36] seconds. The task execution efficiency $\lambda_{i,j}$ is a random number in [0.1, 1], which is similar to the setup in [36]. The coefficient $A_{j,k}$ of the kth resource on server S_j, is a random number in [0.5, 1]. The number of the kth resource on server S_j, i.e., $B_{j,k}$, is a random number in [80, 100]. Similar to [17], we set the power model constant $\alpha_{i,j}$ of processing T_i to be a random number in $\{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. With the above parameters, we can decide the energy consumption-optimal schedule for any application by Algorithm 1. Suppose that the completion time with this schedule is $\Re$, then we know that for any application whose real-time requirement $\mathcal{R}$ is greater than $\Re$, the schedule from Algorithm 1 is our solution for the energy minimization. In order to effectively evaluate the influence of real-time constraint on the performance of our proposed scheduling algorithms, we need to investigate the timing requirements that are smaller than $\Re$. Thus, we set $\mathcal{R} = \mu \cdot \Re$ where μ is a timing ratio that $\mu \in (0, 1)$.

6.2 Compared Approaches

To shorten an application's completion time, we can either completely rely on task placement adjustment or combine it with the start time adjustment in a backward way as presented in Section 5.3. We separately evaluate the two approaches. In order to meet the real-time requirement, we can also apply a heuristic approach called enhancement heterogeneous earliest finish time (E-HEFT) in [41] that minimizes the makespan of an application with a well balanced load. In addition, we develop a genetic algorithm to optimize the schedule with the aim of minimizing energy consumption. Because we have already derived a time-optimal resource allocation plan after tasks have been placed, we only need to search a task placement plan and start time plan in genetic algorithm (GA). Hardware resources are allocated based on Theorem 2 after deciding a task placement plan.

- TPA: The approach that first applies the energy consumption-optimal schedule of Algorithm 1 to initially find a schedule, based on which the task placement adjustment of Algorithm 2 is applied to minimize the schedule under the real-time constraint.
- TPA-STA: On top of the schedule from TPA, this approach applies the schedule adjustment in a backward way as shown in Algorithm 4 to further reduce the energy consumption.
- E-HEFT: By using the upward rank value of [42] to sort a task allocation order, this approach successively allocates a task to a server and creates a virtual machine with full hardware resources to execute it.



(a) Gaussian elimination with $\rho = 5$ (b) Fast Fourier transform with $\rho = 4$

Fig. 5. Two real-case parallel workflows.

For each task, we calculate its completion time on each server, and allocates it to a server that has the minimum completion time.

- GA: This approach applies GA to find a schedule. The task placement plan and start time plan are encoded in this algorithm. When a real-time constraint is not met, there is a big penalty such that the search abandons the populations that do not meet the timing requirement.

For all compared approaches, we use three metrics to evaluate their performance. The first one is energy consumption. The second one is the success rate to find a schedule, and the last one is the computation time of finding a schedule.

6.3 Results

Gaussian elimination and fast Fourier transform have been widely used as two benchmarks to evaluate the scheduling performance in cloud computing systems [42], [43]. We first present the evaluation results on them, and then the results on synthetic applications. Synthetic applications composed of partially ordered task graphs are generated by using the approach from [44].

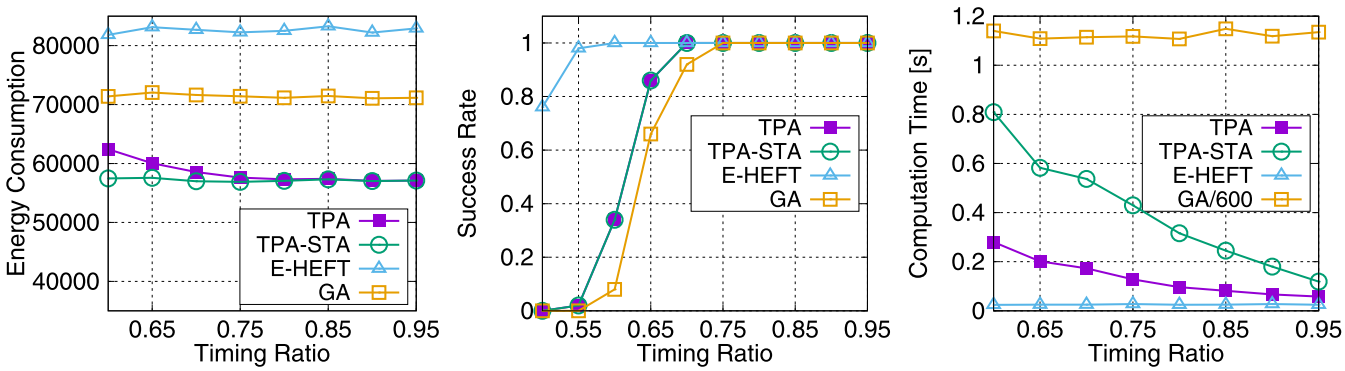
6.3.1 Gaussian Elimination

Gaussian elimination is composed of tasks in the shape as shown in Fig. 5a. Because there is only a single task at its odd level, this application has low parallelism. Its size can be scaled by ρ. Its task count with respect to ρ in this application is $N = \frac{\rho^2+\rho-2}{2}$. Here we mainly investigate the effect of real-time requirement on the scheduling performance of all compared approaches. We set the server count $M = 24$ and the application size $\rho = 24$, and vary the timing ratio μ from 0.5 to 0.95 at a step of 0.05. In this setting, 299 tasks need to be scheduled on 24 servers. We generate 50 applications and present their average energy consumption and computation time for evaluations. The success rate denotes the percentage of applications whose feasible schedules have been successfully found among those 50 applications.

The results are presented in Fig. 6, where Fig. 6a shows that TPA-STA achieves the least energy consumption. The energy consumption of TPA is close to that of TPA-STA, and with the timing ratio increment, the gap between these two approaches become smaller. This is because TPA completely rely on paying extra energy cost to reduce an application's completion time. When the real-time constraint is greatly smaller than $\Re$, more energy cost needs to be paid to reduce completion time. Fig. 6b shows that TPA and TPA-STA achieve the same success rate. This illustrates that compared to a start time adjustment strategy, the task placement adjustment strategy is more powerful to reduce application's completion time. The computation time spent on finding a feasible schedule is presented in Fig. 6c. It shows that both TPA and TPA-STA can find a solution in less than 1 second. Compared to TPA, TPA-STA needs more computation time. This is because TPA-STA goes through all processes of TPA and needs extra computation to adjust the task start time. The computation time decreases with the timing ratio increment, because fewer tasks need to be moved to other servers when the gap between $\Re$ and timing constraint becomes small. As a comparison, GA performs the worst in all three metrics. Especially in the last metric, GA needs 600 times more computation time than TPA and TPA-STA to get a schedule. It should be noted that, because the success rate is very low when $\mu \leq 0.55$, their results of energy consumption and computation time are not listed in Figs. 6a and 6c. In addition, we find that E-HEFT has a very high success rate and its computation time is the least among the four compared approaches. This is because E-HEFT is a lightweight heuristic method that aims to find the minimum completion time. Its energy consumption is the largest because it does not take the energy saving into account.

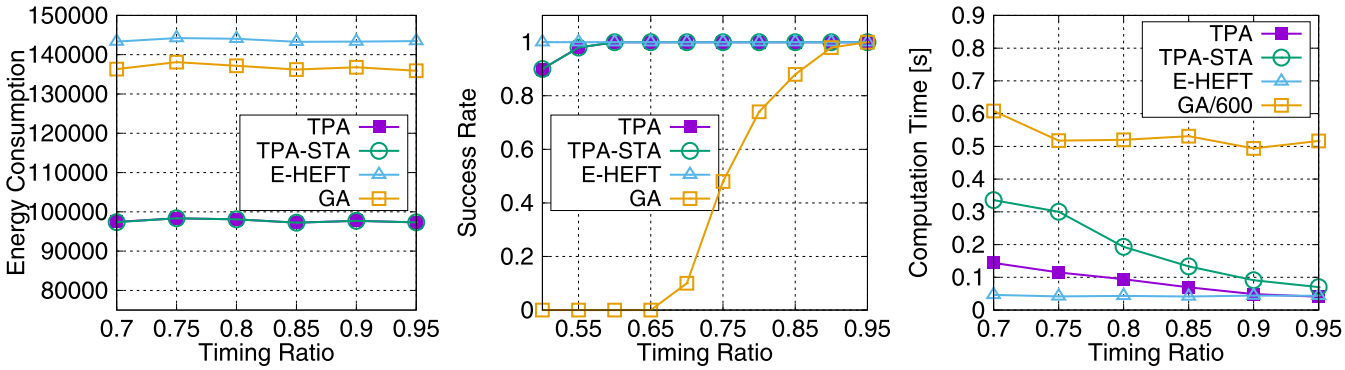
6.3.2 Fast Fourier Transform

Fast Fourier transform is composed of tasks in the shape as shown in Fig. 5b. The size of this application depends on ρ. There are $2\rho - 1$ recursive call tasks and $\rho \log_2 \rho$ butterfly



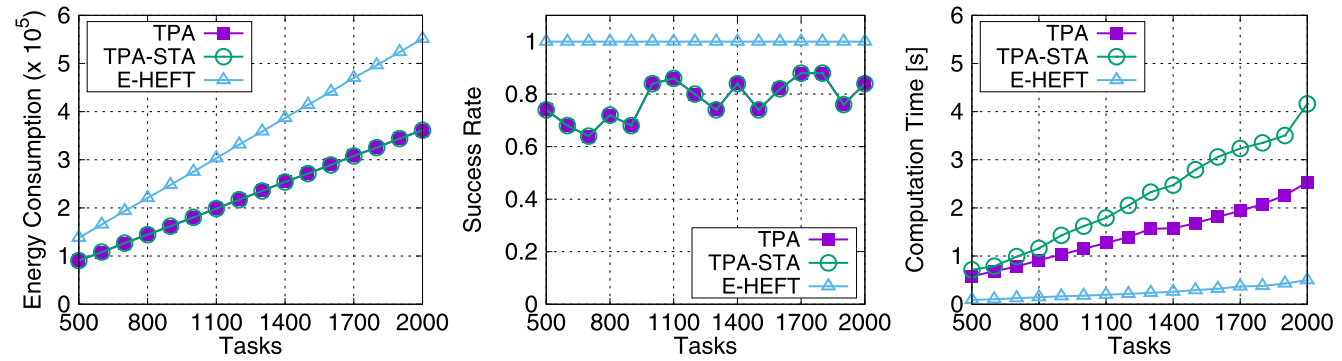
(a) Energy consumption w.r.t., timing ratio (b) Success rate w.r.t., timing ratio (c) Computation time w.r.t., timing ratio

Fig. 6. Results of compared approaches on the Gaussian elimination application.



(a) Energy consumption w.r.t., timing ratio (b) Success rate w.r.t., timing ratio (c) Computation time w.r.t., timing ratio

Fig. 7. Results of compared approaches on the fast Fourier transform application.



(a) Energy consumption w.r.t., tasks (b) Success rate w.r.t., tasks (c) Computation time w.r.t., tasks

Fig. 8. Results of compared approaches on the synthetic application.

operation tasks, where $\rho = 2^k, k \in \mathbb{N}^+$. Because there are no sequential tasks between butterfly operation tasks, this application has high parallelism. We also investigate the effect of its real-time requirement on the scheduling performance. We set the server count $M = 24$ and the application size $\rho = 64$, and vary the timing ratio from 0.5 to 0.95 at a step of 0.05, where 511 tasks need to be scheduled on 24 servers. We also generate 50 applications and present their average energy consumption, average computation time, and success rate.

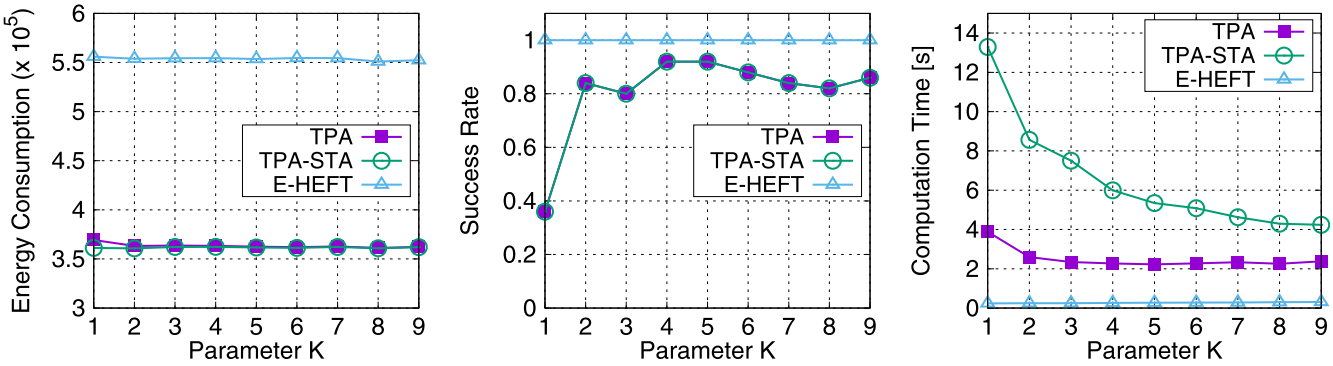
The simulation results are presented in Fig. 7. We observe that the performances of TPA and TPA-STA are very close, and both of them outperform GA to a significant margin in three metrics. It should be noted that, compared to the scheduling results of Gaussian elimination, TPA and TPA-STA have a higher success rate on the scheduling of fast Fourier transform. This is because the structure of fast Fourier transform is more parallel than that of Gaussian elimination, which provides a task placement strategy with more options to reduce completion time.

6.3.3 Synthetic Applications

In order to validate our proposed scheduling techniques on general applications, we now apply them to schedule randomly generated applications. An application is randomly generated in a way similar to [44]'s:

- Total number of tasks: in an application, the total number of tasks N is a number in the range of [500, 2000].
- Task count of each level: at each level of an application, the number of tasks is a random integer within $[10K, 10(K+1)]$, where $K \in \{1, 2, \ldots, 9\}$. The larger K is, the more parallel an application has.
- Task dependency constraint: the task is restricted to communicate with tasks from its neighboring level. Except for the entry and exit tasks, the number of predecessors and successors of any task is an integer within [1, 6].
- Other parameters: the number of servers is set to 50, and other parameters such as task load and cloud computing platform are the same as described in the above subsection.

We mainly investigate the effect of N and K on the scheduling performance. By setting the timing ratio as 0.55 and $K = 9$ and varying N from 500 to 2,000, we get the energy consumption per application, success rate, and computation time per application as presented in Fig. 8. We do not present the results of GA because it is difficult for GA to find a schedule in this setting. We find that the energy consumption of TPA and TPA-STA increases slower than that of E-HEFT with N. This is because TPA and TPA-STA choose the energy consumption-optimal server to execute a task, while E-HEFT adopts the time-optimal server to execute one. The time-optimal server consumes more energy than the energy consumption-optimal server. Besides, we find that N does not have a certain impact on the success rate of TPA and TPA-STA to find a schedule. In addition, we find that the computation time of TPA and TPA-STA increases faster than that of E-HEFT. The reason is that there are more levels in an application when N is large. More levels mean that there are more key bottleneck tasks to be



(a) Energy consumption w.r.t., parameter K (b) Success rate w.r.t., parameter K (c) Computation time w.r.t., parameter K

Fig. 9. Results of compared approaches on the synthetic application.

handled, thus leading to more computation. On the other side, E-HEFT does not rely on such levels to schedule tasks, and its energy consumption only linearly increases with N.

By setting the timing ratio as 0.55 and N as 2,000 and varying K from 1 to 9, we investigate the effect of task concurrency on their scheduling performance. In this setting, it is also difficult for GA to find a schedule. Thus, its results are not listed. As presented in Fig. 9, we find that TPA-STA is slightly better than TPA in reducing energy consumption when K is small. This means that TPA-STA has a small advantage for handling a low-parallel application. This conclusion is also confirmed by the results from processing Gaussian application as shown in Fig. 6a. Regarding to the success rate, we find that larger K improves the success rate of TPA and TPA-STA, which means that TPA and TPA-STA are more effective to find a schedule for an application with high task concurrency. This conclusion can also be confirmed by comparing the success rate of scheduling fast Fourier transform and Gaussian elimination. In addition, we find that the computation time of TPA and TPA-STA decreases as K increases, because the number of levels is reduced with more tasks to be grouped into a level. On the contrary, the computation time of E-HEFT keeps constant because N does not change.

7 Conclusion

Finding a high-performance schedule for cloud computing is important to reduce a system's energy consumption. In this paper, we derive an energy consumption-optimal solution for task placement in a parallel application, and then derive a time-optimal resource allocation plan for independent tasks at the same level. Based on the two closed-form solutions, we propose a heuristic algorithm TPA to adjust task placements, and propose a heuristic algorithm TPA-STA to combine the task placement adjustment with task start time adjustment in a backward way. Experimental results show that our proposed approaches need far less computation time to find a good schedule for a large-size parallel application in cloud than a meta-heuristic genetic algorithm. Thus, the presented approaches in this paper have greater potential in handling a cloud scheduling problem, especially for large-size parallel applications. However, it should be noted that our current work only considers real-time requirements in scheduling, which may not be enough in reality. Our future work is thus to design more applicable algorithms that can cover the constraints of more performance indices such as reliability and throughput.

Acknowledgments

This work has partly been supported in part by the National Natural Science Foundation of China under Grant 61802013, in part by the Beijing Leading Talents Program under Grant Z191100006119031, in part by the Talent Foundation of Beijing University of Chemical University under Grant buctrc201811, in part by the Fundamental Research Funds for the Central Universities under Grant XK1802-4, and in part by the Deanship of Scientific Research (DSR) at King Abdulaziz University, Jeddah, under grant no. KEP-2-135-39.

References

[1] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and Internet of Things: A survey," *Future Gener. Comput. Syst.*, vol. 56, pp. 684–700, 2016.

[2] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless Commun. Mobile Comput.*, vol. 13, no. 18, pp. 1587–1611, 2013.

[3] M. H. Ghahramani, M. C. Zhou, and T. H. Chi, "Toward cloud computing QoS architecture: Analysis of cloud systems and cloud services," *IEEE/CAA J. Automatica Sinica*, vol. 4, no. 1, pp. 6–18, Jan. 2017.

[4] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[5] K. Li, "Power and performance management for parallel computations in clouds and data centers," *J. Comput. Syst. Sci.*, vol. 82, no. 2, pp. 174–190, 2016.

[6] J. Bi et al., "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 2, pp. 1172–1184, Apr. 2017.

[7] H. Yuan, J. Bi, W. Tan, M. Zhou, and J. Li, "TTSA: An effective scheduling approach for delay bounded tasks in hybrid clouds," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3658–3668, Nov. 2017.

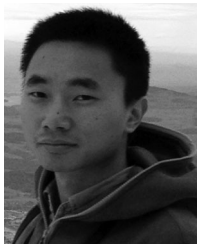
[8] H. Yuan, J. Bi, W. Tan, M. Zhou, and K. Sedraoui, "WARM: Workload-aware multi-application task scheduling for revenue maximization in SDN-based cloud data center," *IEEE Access*, vol. 6, pp. 645–657, 2018.

[9] H. Yuan, J. Bi, W. Tan, M. Zhou, and A. Ammari, "Time-aware multi-application task scheduling with guaranteed delay constraints in green data center," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 3, pp. 1138–1151, Jul. 2018.

[10] H. Yuan, J. Bi, and M. Zhou, "Profit-sensitive spatial scheduling of multi-application tasks in distributed green clouds," *IEEE Trans. Autom. Sci. Eng.*, pp. 1–10, to be published, doi: 10.1109/TASE.2019.2909866.

[11] Y. Song, X. He, Z. Liu, W. He, C. Sun, and F.-Y. Wang, "Parallel control of distributed parameter systems," *IEEE Trans. Cybern.*, vol. 48, no. 12, pp. 3291–3301, Dec. 2018.

[12] K. Katoh, K. Misawa, K.-I. Kuma, and T. Miyata, "MAFFT: A novel method for rapid multiple sequence alignment based on fast fourier transform," *Nucleic Acids Res.*, vol. 30, no. 14, pp. 3059–3066, 2002.
[13] G. Xie, G. Zeng, R. Li, and K. Li, "Energy-aware processor merging algorithms for deadline constrained parallel applications in heterogeneous cloud computing," *IEEE Trans. Sustain. Comput.*, vol. 2, no. 2, pp. 62–75, Apr.–Jun. 2017.
[14] K. Huang, B. Hu, J. Botsch, N. Madduri, and A. Knoll, "A scalable lane detection algorithm on COTSs with OpenCL," in *Proc. Conf. Des. Autom. Test Europe*, 2016, pp. 229–232.
[15] X. Wang, C. Kiwus, C. Wu, B. Hu, K. Huang, and A. Knoll, "Implementing and parallelizing real-time lane detection on heterogeneous platforms," in *Proc. IEEE Conf. Appl.-Specific Syst. Archit. Processors*, 2018, pp. 1–8.
[16] J. D. Ullman, *NP-Complete Scheduling Problems*. Cambridge, MA, USA: Academic Press, Inc., 1975.
[17] L. Shi, Z. Zhang, and T. Robertazzi, "Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1607–1620, Jun. 2017.
[18] Q. Zhao, C. Xiong, C. Yu, C. Zhang, and X. Zhao, "A new energy-aware task scheduling method for data-intensive applications in the cloud," *J. Netw. Comput. Appl.*, vol. 59, pp. 14–27, 2016.
[19] S. Hosseinimotlagh, F. Khunjush, and R. Samadzadeh, "SEATS: Smart energy-aware task scheduling in real-time cloud computing," *The J. Supercomputing*, vol. 71, no. 1, pp. 45–66, 2015.
[20] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.
[21] H. Yuan, J. Bi, and M. Zhou, "Temporal task scheduling of multiple delay-constrained applications in green hybrid cloud," *IEEE Trans. Services Comput.*, p. 1, to be published, doi: 10.1109/TSC.2018.2878561.
[22] H. Yuan, B. Jing, and M. C. Zhou, "Spatial task scheduling for cost minimization in distributed green cloud data centers," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, pp. 729–740, Apr. 2019.
[23] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 4, pp. 875–887, Jul. 2019.
[24] Y. Hou, N. Q. Wu, M. C. Zhou, and Z. W. Li, "Pareto-optimization for scheduling of crude oil operations in refinery via genetic algorithm," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 47, no. 3, pp. 517–530, Mar. 2017.
[25] Y. Cao, H. Zhang, W. Li, M. Zhou, Y. Zhang, and W.-A. Chaovalitwongse, "Comprehensive learning particle swarm optimization algorithm with local search for multimodal functions," *IEEE Trans. Evol. Comput.*, vol. 23, no. 4, pp. 718–731, Aug. 2019.
[26] W. Dong and M. C. Zhou, "A supervised learning and control method to improve particle swarm optimization algorithms," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 47, no. 7, pp. 1135–1148, Jul. 2017.
[27] Y. Yu, S. Gao, Y. Wang, and Y. Todo, "Global optimum-based search differential evolution," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 2, pp. 379–394, Mar. 2019.
[28] Y. Shen, Z. Bao, X. Qin, and J. Shen, "Adaptive task scheduling strategy in cloud: When energy consumption meets performance guarantee," *World Wide Web*, vol. 20, no. 2, pp. 155–173, 2017.
[29] L. Zuo, S. Lei, S. Dong, C. Zhu, and T. Hara, "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing," *IEEE Access*, vol. 3, pp. 2687–2699, 2017.
[30] G. N. Gan, T. L. Huang, and G. Shuai, "Genetic simulated annealing algorithm for task scheduling based on cloud computing environment," in *Proc. Int. Conf. Intell. Comput. Integr. Syst.*, 2010, pp. 60–63.
[31] R. Jena, "Energy efficient task scheduling in cloud environment," *Energy Procedia*, vol. 141, pp. 222–227, 2017.
[32] X. Zhen, W. Song, and C. Qi, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.
[33] T. Baker, B. Aldawsari, M. Asim, H. Tawfik, Z. Maamar, and R. Buyya, "Cloud-senergy: A bin-packing based multi-cloud service broker for energy efficient composition and execution of data-intensive applications," *Sustain. Comput. Informat. Syst.*, vol. 19, pp. 242–252, 2018.
[34] P. Y. Zhang and M. C. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 772–783, Apr. 2018.
[35] M. Al-khafajiy, T. Baker, A. Waraich, D. Al-Jumeily, and A. Hussain, "IoT-fog optimal workload via fog offloading," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput. Companion*, 2018, pp. 359–364.
[36] D. Li and J. Wu, "Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 810–823, Mar. 2015.
[37] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 168–180, Apr.–Jun. 2014.
[38] N. Kim, J. Cho, and E. Seo, "Energy-credit scheduler: An energy-aware virtual machine scheduler for cloud systems," *Future Gener. Comput. Syst.*, vol. 32, no. 1, pp. 128–137, 2014.
[39] C. Gu, H. Huang, and X. Jia, "Power metering for virtual machine in cloud computing-challenges and opportunities," *IEEE Access*, vol. 2, pp. 1106–1116, 2014.
[40] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 39–50.
[41] Y. Samadi, M. Zbakh, and C. Tadonki, "E-heft: Enhancement heterogeneous earliest finish time algorithm for task scheduling based on load balancing in cloud computing," in *Proc. IEEE Int. Conf. High Perform. Comput. Simul.*, 2018, pp. 601–609.
[42] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[43] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, and X. Huang, "Enhanced energy-efficient scheduling for parallel applications in cloud," in *Proc. IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2012, pp. 781–786.
[44] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *Proc. IEEE Int. Workshop Hardware/Software Codesign*, 1998, pp. 97–101.



Biao Hu received the BSc degree in control science and engineering from the Harbin Institute of Technology, in 2010, the MSc degree in control science and engineering from Tsinghua University, in 2013, and the PhD degree from the Department of Computer Science, Department of Informatics of the Technische Universität Müchen, in 2017. He is currently an associate professor with the College of Information Science and Technology, Beijing University of Chemical Technology. His research interests include the path planning of autonomous driving, and OpenCL computing in heterogeneous system. He is a member of the IEEE.

Zhengcai Cao received the MS and PhD degrees from the Harbin Institute of Technology, Harbin, China, in 2001 and 2005, respectively. During January 2006–June 2008, he was a postdoctoral research at the Research Center of Control Science and Engineering, Tongji University. From 2008, he became a faculty member of Beijing University of Chemical Technology, Beijing, China, where he is currently a professor. From March 2014 to March 2015, he was a visiting scholar at King's College London, United Kingdom. His current research interests include system engineering and intelligent control. He is a member of the IEEE.

Mengchu Zhou (S'88-M'90-SM'93-F'03) received the BS degree in control engineering from the Nanjing University of Science and Technology, Nanjing, China, in 1983, the MS degree in automatic control from the Beijing Institute of Technology, Beijing, China, in 1986, and the PhD degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1990. He joined New Jersey Institute of Technology (NJIT), Newark, NJ, in 1990, and is now a distinguished professor of Electrical and Computer Engineering. His research interests include Petri nets, intelligent automation, Internet of Things, big data, web services, and intelligent transportation. He has more than 800 publications including 12 books, more than 500 journal papers (more than 400 in IEEE transactions), 12 patents and 29 book-chapters.