ET2FA: A Hybrid Heuristic Algorithm for Deadline-constrained Workflow Scheduling in Cloud

Zaixing Sun, Boyu Zhang, Chonglin Gu, Ruitao Xie, Bin Qian, and Hejiao Huang

Abstract—Cloud computing is an emerging computational infrastructure for cost-efficient workflow execution that provides flexible and dynamically scalable computing resources at pay-as-you-go pricing. Workflow scheduling, as a typical NP-Complete problem, is one of the major issues in cloud computing. However, in the cloud scenario with unlimited resources, how to generate an efficient and economical workflow scheduling scheme under the deadline constraint is still an extraordinary challenge. In this paper, we propose a hybrid heuristic algorithm called enhanced task type first algorithm (ET2FA) to solve deadline-constrained workflow scheduling in cloud with new features such as hibernation and per-second billing. The objectives to be minimized include the total cost and total idle rate. ET2FA involves three phases: 1) Task type first algorithm, which schedules tasks based on topological level and task types, and utilizes a compact-scheduling-condition based VM selection method to assign each task. 2) Delay operation based on block structure, which further optimizes total cost and total idle rate based on block structure properties. 3) Instance hibernate scheduling heuristic, which sets an instance to hibernate if idle for a duration. Extensive simulation experiments based on seven well-known real-world workflow applications show that ET2FA delivers better performance in comparison to the state-of-the-art algorithms.

Index Terms—Workflow scheduling, cloud computing, deadline constraint, directed acyclic graph, hibernate instance

1 Introduction

Cloud computing is emerging as a primary computing paradigm that is researched, developed, and deployed by academia, industry and government in recent years [1], [2]. In a cloud platform, the computing resources are heterogeneous, elastic, and almost unlimited, which can be leased at any time in a pay-as-you-go service model [3]. Recently, some cloud service platforms, such as AWS EC2, Google Cloud, Microsoft Azure, etc., have introduced instance hibernate (suspended or deallocated) function in a lifecycle of an instance, which can save the instance booting time and rental expenses. In addition, they also support per-second billing, which brings customers closer to being billed ONLY for the time when resources are actually used. These two measures will enable customers to make full use of the elasticity of cloud computing to save costs.

Workflow has been proved to be an efficient and popular paradigm to model various scientific computing problems with massive amounts of data and complex constraints in various fields, such as astronomy, bioinformatics, and physics [3], [4]. It is usually described by directed acyclic graph (DAG), in which nodes represent application tasks, and directed edges represent inter-task data dependencies [5]. Based on its powerful computing capability, the infrastructure as a service (IaaS) cloud offers users a new utility-based platform to execute large scale workflows [6]. Customers can execute their workflows by renting resources from cloud service providers. The essence of workflow scheduling is to establish a set of effective mapping relationships from tasks to virtual machines (VMs) in cloud to minimize makespan or monetary cost under the constraints of Quality of Service (QoS, such as deadline), so as to achieve efficient utilization and balanced allocation of system resources.

Workflow scheduling makes the following decisions or trade-offs: (1) Determining the scheduling sequence of tasks. Workflow tasks contain dependency constraints. Although tasks can be divided into topological levels and scheduled in turn according to the levels, the scheduling of tasks in the same level can be reduced to the bag-of-tasks scheduling problem, which is still NP-Hard. (2) Selecting VM. To execute a workflow at low cost, the low-cost VM will be selected. In this way, the task takes long time to execute, which may result in a deadline miss. To finish the workflow as early as possible, the VM with high computing power will be selected. In this way, the task has short execution time, and it is easy to meet deadline, but with high cost. In addition, when more VMs are used, tasks are scattered over VMs, increasing data transmission time among tasks and the idle time of VMs. Therefore, for VM selection, we should not only choose the appropriate VM type to meet deadline with low cost, but also avoid using too many VMs in order to reduce data transmission time and idle time.

Since workflow scheduling problem is NP-Complete

- *Z. Sun, C. Gu and H. Huang, are with the School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen 518000, China, and also with Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, Shenzhen 518000, China. E-mail: szx_1010@stu.hit.edu.cn, {guchonglin, huanghejiao}@hit.edu.cn.*
- *B. Zhang is with School of Artificial Intelligence, Changchun University of Science and Technology, Changchun 130000, China. E-mail: 2583392480@qq.com.*
- *R. Xie is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518000, China. E-mail: drtxie@gmail.com.*
- *B. Qian is with the School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China. E-mail: bin.qian@vip.163.com.*

*Manuscript received *; revised *; accepted *. (Corresponding author: Chonglin Gu.)*

[7]–[9], its solution methods [10] mainly include heuristic algorithms [7], [11]–[14], meta-heuristic algorithms [4], [15] and artificial intelligence algorithms [16], [17]. Heuristic algorithms depend on the nature of the problem, and are designed according to the characteristics of the problem. Meta-heuristic algorithms and artificial intelligence algorithms are usually independent of the problem, and perform iterative optimization through a certain evolution mechanism [18]. When selecting resources for tasks, these algorithms usually randomly select resources without guidance. The existing algorithms usually need to adjust the control parameters manually, and run for a long time due to their slow convergence speed. Moreover, when the execution scenario has new features, the original scheduling algorithm is no longer applicable. In this paper, we consider a cloud environment with new features of hibernation and per-second billing.

Aiming at the significance of workflow scheduling in cloud and the deficiency of current algorithms, this paper proposes a hybrid heuristic algorithm called enhanced task type first algorithm (ET2FA) for deadline-constrained workflow scheduling in cloud to minimize the total cost and total idle rate. In cloud, the billing method is per-second billing with a minimum of 60 seconds. The heterogeneous VM instances are acquired and released dynamically, and they can also be hibernated. VMs with different configurations have different bandwidths. The main contributions of this paper are as follows:

- A cloud-based workflow scheduling model is established, which rents VM instances in a per-second mode while considering hibernating the idle VMs at a much lower price.
- We creatively propose a task scheduling algorithm based on topological level. Within each level, we prioritize the tasks according to the workflow structure. When assigning each task, we devise a compact-scheduling-condition based VM selection method, which can reduce data transmission time and idle time.
- We theoretically prove a determination condition that can simultaneously save cost and improve resource utilization by analyzing the property of the block structure (a sequence of tasks are continuously executed without idle intervals on the same VM), and then propose a delay operation based on block structure to further optimize total cost and total idle rate.
- By simulation experiments with seven well-known real-world workflow applications, the proposed algorithm is verified to outperform five baseline algorithms (including two heuristic algorithms, two meta-heuristic algorithms and a reinforcement learning algorithm), in total cost, total idle rate and running time of algorithms.

A preliminary result of our work was presented at the conference IEEE CLOUD 2021 [19]. Compared with previous work, this paper includes significant new contents: 1) A new cloud resource model is considered, which includes instance hibernation mode and per-second billing with a minimum of 60 seconds, and does not limit the number of instances. 2) During VM selection for each task, we preferentially select the VMs with running tasks at current topological level and then its upper level. This can make the scheduling more compact. 3) To further optimize total cost and total idle rate, delay operation based on block structure is proposed by analyzing the property of block structures. 4) Instance hibernate scheduling heuristic is designed to hibernate instance if idle for a duration.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the cloud workflow scheduling model, including resource model, workflow application model, deadline model and workflow scheduling. Section 4 provides details of ET2FA to address this problem. Section 5 presents the experimentation and evaluation. Finally, Section 6 concludes the paper.

2 Related Work

2.1 Workflow Scheduling Problem in Cloud

A substantial number of research efforts have been devoted towards solving the cost and maximum completion time issues of workflow scheduling. According to different execution scenarios, the workflow scheduling problem has different characteristics. The execution scenarios (resource models) are evolving from homogeneous single-core processors to multicore processors with heterogeneous resources.

In **homogeneous** resource environment, all computing resources have the same configuration of CPU, memory, etc. Byun et al. [20] proposed a partitioned balanced time scheduling algorithm, which estimates the minimum number of computing hosts required to execute a workflow under user deadline, minimizing the financial cost during the entire application lifetime. Wu et al. [11] proposed a two-stage method minimal slack time and minimal distance algorithm and VM instance hour minimization for deadline constrained DAG applications deployed on cloud.

In **heterogeneous** scenes, Abrishami et al. [12] proposed IaaS Cloud Partial Critical Paths (IC-PCP) for minimizing the cost of workflow execution under deadline constraints. IC-PCP constructed critical paths of the service processes, and the tasks on each critical path are assigned to a cheapest VM that satisfies the deadline constraints. Rodriguez et al. [4] considered VM boot time and developed particle swarm optimization (PSO) to minimize overall workflow execution cost while meeting the deadline constraint in clouds. However, they mainly designed the resource provisioning and scheduling strategy, and there was no improvement on the PSO. Sahni et al. [13] considered the VM performance variability and instance acquisition delay, and proposed Just-in-Time (JIT-C) algorithm to minimize cost of workflow execution under deadline constraints. In JIT-C algorithm, tasks with serial characteristics are merged, which reduces the cost of data transmission and the complexity of problem solving to a certain extent. Xiao et al. [5] proposed a cooperative coevolution genetic programming (CCGP) algorithm to minimize the makespan. The CCGP algorithm automatically learns two high-level heuristics through genetic programming. Song et al. [1] broke the tradition of atomic tasks and devised a new workflow scheduling model, which modeled the heavy tasks as composite tasks, and assigned multiple service instances to execute a composite task. To solve this problem, they proposed a nested particle swarm optimization algorithm to optimize the scheduling order of tasks and instances respectively. Domanal et al. [21] considered real-time workflow scheduling and presented a novel hybrid

bio-inspired algorithm by integrating the modified particle swarm optimization and modified cat swarm optimization algorithm to the efficient and rapid allocation of resources to the clients. Their algorithm not only reduces the average response time but also increases the resource utilization by approximately 12%. Qin et al. [22] proposed a knowledge-based adaptive discrete water wave optimization (KAD-WWO) algorithm to solve the cost-minimization and deadline constraint cloud workflow scheduling problem.

With the development of **multicore** processor technology, Deldari et al. [23] established a heterogeneous computing resource model, in which each multicore processor consists of several homogeneous cores, and proposed a cluster combining algorithm, to minimize the execution cost while meeting the deadline constraints submitted by users. Zhu et al. [24] considered both the multiprogrammed use of computing resources on heterogeneous IaaS platforms and the multi-resource demands of tasks, and proposed a new list-scheduling framework. To make full use of cloud resources, this framework can efficiently pack tasks onto VMs and support the dynamic expansion of VMs in the scheduling process. Based on this framework, a deadline-constrained workflow scheduling algorithm was proposed to minimize the cost of workflow execution.

Most of the aforementioned studies focus on conventional cloud environment, and a few studies involve VM boot time. In fact, some characteristics, such as data transmission and startup time, can't be ignored. This paper studies a more realistic scenario, which considers heterogeneous resources with unlimited number, VM boot time, VM bandwidth and hibernate function, etc.

2.2 Workflow Scheduling Algorithm in Cloud

The existing **heuristic algorithms** for the cloud workflow scheduling mainly include the following: Heterogeneous Earliest Finish Time (HEFT) [7], [25], Load Balancing Techniques [26], Priority or Deadline Based Scheduling Algorithm [27] and other algorithms [11]–[14]. Heuristic algorithms are suitable for problems with rules to follow, such as inter-task data dependencies in workflow.

Due to its advantages in convergence speed and accuracy, **Meta-heuristic algorithm** is one of the common strategies to solve NP-Hard optimization problems. At present, the common meta-heuristic algorithms for solving workflow scheduling problems mainly include Particle Swarm Optimization [4], [28], Ant Colony Optimization [6], [15], [29], HYBRID bio-Inspired algorithm [21], Squirrel Search Algorithm [30], Genetic Algorithm [31], etc. These algorithms overcome the shortcomings of traditional analytical algorithms to a great extent, and provide new ideas and means for solving scheduling problems. Further research on this kind of technology and its better application in solving scheduling problems will undoubtedly have a positive impact on the development of scheduling technology and other constrained combinatorial optimization problems.

Artificial intelligence algorithm has also been studied in the field of workflow scheduling in cloud environment in the last few years, such as Q-Learning [16], [32], Artificial Neural Network [33], Bayesian Network [17] and so on. For example, Zhao et al. [16] proposed QL-HEFT which combines Q-Learning with HEFT. The QL-HEFT utilizes upward ranking values from HEFT which are used for reward in Q-learning process. The algorithm sorts the tasks according to the convergent Q-table, and assigns the tasks to the VMs based on the earliest finish time strategy.

Theoretically, the widespread application and development of workflow scheduling technology depends not only on the improvement and development of various heuristic algorithm technologies based on natural laws, but also on the deep understanding and research of scheduling domain knowledge, so as to organically combine the algorithm and prior domain knowledge of the problem to achieve global optimization. Therefore, it is of great significance to study the scheduling problem theoretically for developing optimization technology and solving complex combinatorial optimization problems.

3 Cloud Workflow Scheduling Model

In this section, the resource model and workflow application model are described, and then the workflow scheduling model is established. To facilitate reading, the symbols and variables commonly used in this paper are listed in Table 1.

Table 1
Symbols and Meanings.

Symbol	Definition
Constants	
v_h^P	The instance v_h's type.
v_h^U	The instance v_h's processing capacity.
v_h^M	The instance v_h's per-unit price.
a_i	A task in the DAG.
w_i	The computation of task a_i.
d_{ij}	The amount of data to be transferred from task a_i to task a_j.
$Suc(a_i)$	The sets of all direct successors of task a_i.
$Pre(a_i)$	The sets of all direct predecessors of task a_i.
Variables	
t_{ih}^E	The execution time of a task a_i on VM v_h.
v_i^A	The VM where task a_i is executed.
t_i^S	The actual start time of task a_i.
t_i^F	The actual finish time of task a_i.
$\tilde{t}_h^S$	The lease start time of instance v_h.
$\tilde{t}_h^E$	The lease end time of instance v_h.
$\tilde{t}_{ih}^A$	The available start time of task a_i on VM v_h.
V^C	The VM set with running tasks at current topological level.
V^P	The VM set with running tasks at immediately preceding topological level.

3.1 Resource Model

3.1.1 Resource Configurations

Cloud service providers deliver computing resources to customers at different prices via heterogeneous VM instances with various of CPU, storage, and network bandwidth, without limiting the number of VMs. Let $P = \{p_k|k = 1, 2, \cdots, m\}$ represent the set of all instance types, where m is the total number of types. VM instances have the following characteristics:

- $U = \{U(p_k)|p_k \in P\}$ is the set of processing capacity of CPU in Giga Floating Point Operations Per Second (GFLOPS, a widely used metric [4], [13], [32]), where $U(p_k)$ is the processing capacity of instance type p_k. The

higher the processing capacity of CPU, the shorter the execution time of its task.

- $B = \{B(p_k, p_h)|p_k, p_h \in P\}$ is the set of communication bandwidth between different instance types. $B(p_k, p_h)$ is the communication bandwidth between instance types p_k and p_h, which depends on the smaller bandwidth of the two instances (denoted as $b(p_k)$ and $b(p_h)$, respectively) [34], [35]. That is, $B(p_k, p_h) = \min\{b(p_k), b(p_h)\}$.
- $M = \{M(p_k)|p_k \in P\}$ is the set of leasing prices, where $M(p_k)$ is the per-unit price of instance type p_k.

Let $V = \{v_1, v_2, \cdots, |V|\}$ represent the VM instances leased by a customer, where $|V|$ is the total number of VMs. $v_h^P = p_k$ represents instance v_h's type. $v_h^U = U(v_h^P)$ represents instance v_h's processing capacity. $v_h^M = M(v_h^P)$ represents instance v_h's per-unit price.

The pricing model is based on a pay-as-you-go billing scheme and the users are charged for the number of time intervals for the instances they lease, even if the last time interval is not fully used. The time interval is the minimum billing period. It is determined by the service policy of a cloud service platform. In this paper, the time interval is 1 second with a mandatory-minimum of 60 seconds whenever an instance switches to a new state. For example, the billing time of the period $[t_1, t_2], t_1 < t_2$ is,

$$g(t_1, t_2) = \lceil \max\{(t_2 - t_1), 60\} \rceil . \quad (1)$$

In reality, cloud providers charge storage services for storing data files according to the allocated capacity, but these costs are not accounted for in the resource model since they are independent of the scheduling algorithms. It is assumed that instances have sufficient RAM for tasks and the CPU capacity is considered as the only factor that determines the execution time of tasks.

3.1.2 Instance Lifecycle

Fig. 1 illustrates the diagrammatic sketch of the uptime segmentation of instance lifecycle. According to the rental unit price, it can be divided into the following two states:

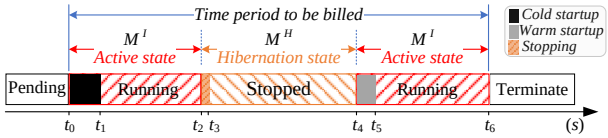


Fig. 1. An illustration of the uptime segmentation of instance lifecycle.

1) *Active state.* When an instance is created, the cloud service provider prepares the operating system and application server specified by user. This time period is called ***pending*** state and not billed. Then the user launches the instance and deploys the execution environment of the workflow. This process is called ***cold startup***, and when the user launches an existing (and stopped) instance, this process is called ***warm startup***. After startup, the instance enters the ***running*** state and the task can be executed. M^I is the per-unit price of the instance in active state (cold/warm startup and running), as shown in Fig. 1.

2) *Hibernation state.* When ***hibernating*** instance, it enters ***stopping*** state, which is the process of transition from running state to hibernation state, and then enters ***stopped*** state. When an instance enters hibernation state, its contents in the instance memory (RAM) will be saved to Elastic Block Store (EBS) root volume. Only EBS volumes and elastic IP are charged in the hibernation state[1]. If needed, an instance can be released (terminated) directly in the running state and no longer charged. M^H is the per-unit price of the instance in hibernation state (stopping and stopped), as shown in Fig. 1.

In a life cycle of an instance, cold startup time $(t_1 - t_0)$, warm startup time $(t_5 - t_4)$ and stopping time $(t_3 - t_2)$ are generally known, while running time and stopped time are determined by the scheduling time of tasks. It is worth noting that a new instance billing period will start again when the state is switched, with 60s minimum charge. Thus, the rental cost of the instance in Fig. 1 is as follows:

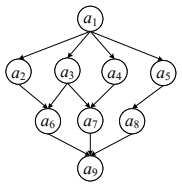
$$cost = \Big(g(t_0, t_2) + g(t_4, t_6)\Big)M^I + g(t_2, t_4)M^H. \quad (2)$$

3.2 Workflow Application Model

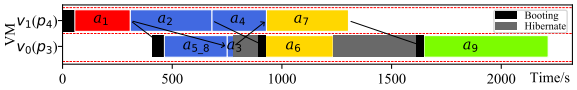
Workflow is represented by directed acyclic graph (DAG) $G = (A, W, E, D)$, which are described as follows:

- $A = \{a_i|i = 1, 2, \cdots, n\}$ is the set of tasks, where a_i is a task in the DAG and n is the total number of tasks.
- $W = \{w_i|i \in A\}$ is the set of weights on tasks, which represents the computation of tasks in giga floating point operations (GFLOP). $t_{ih}^E = \dfrac{w_i}{v_h^U}$ is the execution time of a task a_i on VM v_h.
- $E = \{e_{ij} = (a_i, a_j)|a_i, a_j \in A; i < j\}$ is the set of dependencies between tasks. Dependency (a_i, a_j) refers to a precedence constraint between tasks a_i and a_j. The sets of all direct successors and predecessors of task a_i are denoted as $Suc(a_i) = \{a_j|(a_i, a_j) \in E\}$ and $Pre(a_i) = \{a_j|(a_j, a_i) \in E\}$, respectively.
- $D = \{d_{ij}|e_{ij} \in E\}$ is the set of transmitted data, where d_{ij} represents the amount of data to be transferred from task a_i to task a_j, in GFLOP. Let $t_{ij}^{kh} = \dfrac{d_{ij}}{B(v_k^P, v_h^P)}$ represent the communication time between task a_i and task a_j, where task a_i is executed on VM v_k and task a_j is executed on VM v_h. When $k = h$, the transmission time on the same VM is 0.

A sample workflow is shown in Fig. 2(a). Each node represents a task and each edges represents the dependencies between tasks. The configurations of the nodes and edges are shown in Table 2.



(a) Sample DAG with 9 tasks.



(b) Gantt chart of corresponding task scheduling.

Fig. 2. A simple workflow and its corresponding schedule. In (a) each node represents a task and the edges show the dependencies between tasks. In (b) each task is mapped onto one available VM, and the dependencies between tasks are all satisfied.

1. Since this paper does not consider the memory constraint, only the cost of elastic IP is considered in the hibernation state.

Table 2
The Configurations of Nodes and Edges in Fig. 2(a).

A	W	E	D	E	D
a_1	120204	(a_1,a_2)	443	(a_6,a_9)	1359
a_2	176974	(a_1,a_3)	137	(a_7,a_9)	1034
a_3	6943	(a_1,a_4)	1478	(a_8,a_9)	157
a_4	117952	(a_1,a_5)	466		
a_5	34835	(a_2,a_6)	733		
a_6	74550	(a_3,a_6)	1005		
a_7	1777628	(a_3,a_7)	6943		
a_8	34526	(a_4,a_7)	143		
a_9	136919	(a_5,a_8)	1151		

3.3 Deadline Model

The deadline of workflow is an important constraint for the workflow scheduling problem. If the deadline is relaxed, there is enough slack time to accommodate for the VM acquisition delay and the performance variation. A comprehensive evaluation requires performance analysis on all possible deadlines. Deadline is usually set by the following rule [11], [13]. We use the maximum execution time $\hat{t}_i^E = \max\left\{t_{ih}^E \middle| p_h \in P\right\}$ and the maximum transmission time $\hat{t}_{ij}^C = \max\left\{\frac{d_{ij}}{b_h} \middle| p_h \in P\right\}$ to estimate start and finish time:

$$\tilde{t}_i^S = \begin{cases} 0, & if\ Pre(a_i) = \varnothing, \\ \max\limits_{a_j \in Pre(a_i)} \left\{\tilde{t}_j^F + \hat{t}_{ji}^C\right\}, & otherwise. \end{cases} \tag{3}$$

$$\tilde{t}_j^F = \tilde{t}_j^S + \hat{t}_j^E. \tag{4}$$

The deadline is set to:

$$deadline = \mu \times \max\left\{\tilde{t}_i^F \middle| a_i \in A\right\}, \tag{5}$$

where $\mu \in \{0.8, 1.1, 1.5, 1.8\}$ is deadline factor.

3.4 Workflow Scheduling

Workflow scheduling is to schedule the tasks of a workflow to the VMs on a cloud platform. In essence, workflow scheduling establishes a mapping between the tasks and the VMs. Fig. 2(b) shows a sample schedule generated for the workflow in Fig. 2(a). The VM types are selected in the simulation experiment in Section 5.1.1.

This work focuses on finding a schedule to execute a workflow on an IaaS cloud such that total cost and total idle rate are minimized while meeting the user defined deadline constraint. A schedule is represented as $\Pi = (V^A, T^S, T^F)$ and $R = (\bar{T}^S, \bar{T}^E, \bar{T}^{HS}, \bar{T}^{HE})$ and the objective is $f(\Pi, R) = (total\ cost, total\ idle\ rate)$.

- $V^A = \left\{v_i^A \middle| i \in A\right\}$ is the mapping of the tasks to the VMs, where v_i^A represents the VM where task a_i is executed.
- $T^S = \left\{t_i^S \middle| i \in A\right\}$ and $T^F = \left\{t_i^F \middle| i \in A\right\}$ are the set of actual start time and finish time of tasks, respectively.
- $\bar{T}^S = \left\{\bar{t}_h^S \middle| h \in V\right\}$ and $\bar{T}^E = \left\{\bar{t}_h^E \middle| h \in V\right\}$ are the set of lease start time and lease end time of VMs, respectively.
- $\bar{T}^{HS} = \left\{\bar{t}_{hk}^{HS}\right\}$ and $\bar{T}^{HE} = \left\{\bar{t}_{hk}^{HE}\right\}$ are the set of start and end time of the kth hibernation state of the hth instance, respectively, where $h \in V$, $k = 1, 2, \cdots, \left|\bar{t}_h^{HS}\right|$.

The total cost consists of two parts: the cost of running state (RC) and the cost of hibernation state (HC).

$$RC_h = \begin{cases} v_h^M g(\bar{t}_h^S, \bar{t}_h^E), & \left|\bar{t}_h^{HS}\right| = 0, \\ v_h^M \Big(g(\bar{t}_h^S, \bar{t}_{h1}^{HS}) + \sum_{k=1}^{\left|\bar{t}_h^{HS}\right|-1} g(\bar{t}_{hk}^{HE}, \bar{t}_{h(k+1)}^{HS}) \\ \qquad + g(\bar{t}_{h\left|\bar{t}_h^{HS}\right|}^{HE}, \bar{t}_h^E)\Big), & \left|\bar{t}_h^{HS}\right| \geq 1. \end{cases} \tag{6}$$

$$HC_h = \begin{cases} 0, & \left|\bar{t}_h^{HS}\right| = 0, \\ M^H \sum_{k=1}^{\left|\bar{t}_h^{HS}\right|} g(\bar{t}_{hk}^{HS}, \bar{t}_{hk}^{HE}), & \left|\bar{t}_h^{HS}\right| \geq 1. \end{cases} \tag{7}$$

Finally, the total cost of executing all tasks in a workflow is defined as:

$$total\ cost = \sum_{h=1}^{|V|} (RC_h + HC_h). \tag{8}$$

Although the number of resource is not limited, the leased instance should also be well utilized. The other goal of the scheduling is to reduce the idle time as much as possible, which is measured by total idle rate in Eq.(9). The smaller the value, the less idle the rented instance resources are. In t_{ih}^E, $v_i^A = h$. Moreover, this goal avoids this situation: even if the resource utilization is small, the total resource utilization may be high due to the large scale resources.

$$total\ idle\ rate = \sum_{h=1}^{|V|} \left(1 - \frac{\sum_i t_{ih}^E}{\bar{t}_h^E - \bar{t}_h^S}\right). \tag{9}$$

Based on the previous definitions, the workflow scheduling problem can be formally defined as follows:

$$\textbf{Minimize} \quad total\ cost,\ total\ idle\ rate \tag{10}$$

$$\textbf{subject to} \quad \max\left\{t_i^F \middle| i \in A\right\} \leq deadline, \tag{11}$$

$$t_i^F \leq t_j^S, \quad (a_i, a_j) \in E. \tag{12}$$

4 The Proposed Workflow Scheduling Algorithm

Cost saving can be achieved through three ways: selecting suitable VMs for the tasks, reducing unnecessary idle time and setting idle instances to hibernate. Therefore, a workflow scheduling algorithm named Enhanced Task Type First Algorithm (ET2FA) is proposed, which is a hybrid heuristic algorithm composed of three stages, as described in Table 3.

Table 3
Three Main Phases of ET2FA.

Section	Phase	Description
4.1	T2FA	Establish the mapping of tasks to resources.
4.2	DOBS	Theorem 1 is used to further optimize the results of T2FA to reduce cost and unnecessary idle time.
4.3	IHSH	Determine when and which state the instance should be switched.

4.1 Task Type First Algorithm (T2FA)

In a workflow, there are some tasks with special characteristics. For example, (1) a sequence of tasks with certain structure can be regarded as one task to simplify workflow; (2) the finish time of a task affects all the start time of its subsequent tasks; (3) the finish time of two or more tasks jointly determines the start time of their successors. Considering these particularities, such tasks can be given priority in our scheduling. Compact scheduling and data transmission are the main factors that affect the total cost and total idle rate, which can be adjusted during VM selection. Based on the above analysis, this section proposes T2FA.

4.1.1 Task Topological Level

Given a DAG-based workflow, its task a_i's topological level $Lev(a_i)$ is defined as [11]:

$$Lev(a_i) = \begin{cases} 0, & if\ Pre(a_i) = \varnothing, \\ \max\limits_{a_j \in Pre(a_i)} \{Lev(a_j)\} + 1, & otherwise. \end{cases} \tag{13}$$

Thus, the set $a_{li}^L \in A^L$ of task in each level can be obtained, as shown in Eq. (14).

$$a_l^L = \{a_i | l = Lev(a_i), a_i \in A\}, \tag{14}$$

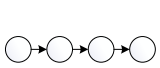
where a_{li}^L represents the ith task in the lth level, $l = 0, 1, \cdots, \max\{Lev\}$, $i = 1, \cdots, |a_l^L|$, and $|a_l^L|$ is the number of tasks in lth level. Especially, tasks at a lower topological level have higher priorities than tasks at a higher level [11].

When there is only one task in a certain level, whether the task can be executed as early as possible plays a key role in the whole workflow scheduling. It should be assigned to VM with higher configuration or earlier completion time. Therefore, it is classified as the 0th type of task (i.e., Eq. (15)), and a scheduling strategy is designed separately for it when scheduling tasks.

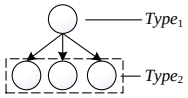
$$Type_0 = \left\{ a_i \middle| a_i \in a_l^L, \left|a_l^L\right| = 1 \right\}. \tag{15}$$

4.1.2 DAG Structure Decomposition

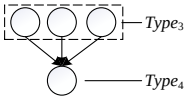
By analyzing the composition characteristics of the upper and lower nodes in the DAG, it can be summarized into the four structures of Fig. 3. The details are as follows.



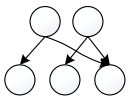
(a) Single output single input.



(b) Multiple output single input.



(c) Single output multiple input.



(d) Multiple output multiple input.

Fig. 3. Four structures in DAG.

- In Fig. 3(a), the structure is single output single input (SOSI), which is a typical serial structure and satisfies the following constraints.

$$|Suc(a_i)| = \sum_{a_j \in Suc(a_i)} |Pre(a_j)| = 1. \tag{16}$$

The best strategy is to assign the tasks to same VM, so they can be merged as a **task block**. The execution time of the **task block** is the sum of its internal task execution time, and the data transmission within the structure is 0. After task merging, DAG can be simplified, and both the solution complexity and space can be reduced.
For example, tasks a_5 and a_8 in Fig. 2(a) are merged into task a_{5_8} in Fig. 2(b).

- In Fig. 3(b), the structure is multiple output single input (MOSI), in which the parent node has multiple child nodes, and the child nodes have a unique parent node. The structure satisfies the following constraints.

$$|Suc(a_i)| = \sum_{a_j \in Suc(a_i)} |Pre(a_j)| > 1. \tag{17}$$

Child nodes are parallel structures, and their start time depends on the unique parent node. To facilitate scheduling, the parent node a_i is defined as the first type of node $Type_1$, child node a_j is defined as the second type of node $Type_2$.

$$Type_1 = \{a_i | a_i \text{ satisfies Eq. (17)}\}, \tag{18}$$

$$Type_2 = \{a_j | a_j \in Suc(a_i), a_i \in Type_1\}. \tag{19}$$

- In Fig. 3(c), the structure is single output multiple input (SOMI), in which the parent nodes have unique child nodes, and the child node has multiple parent nodes. The structure satisfies the following constraints.

$$|Pre(a_j)| = \sum_{a_i \in Pre(a_j)} |Suc(a_i)| > 1. \tag{20}$$

Parent nodes are parallel structures, and their finish time jointly determine the start time of the child node. To facilitate scheduling, the parent node a_i is defined as the third type of node $Type_3$, child node a_j is defined as the fourth type of node $Type_4$.

$$Type_3 = \{a_i | a_i \text{ satisfies Eq. (20)}\}, \tag{21}$$

$$Type_4 = \{a_j | a_j \in Suc(a_i), a_i \in Type_3\}. \tag{22}$$

- In Fig. 3(d), the structure is a general case of multiple output multiple input (MOMI) and is not analyzed separately.

4.1.3 Task Scheduling and VM Selection

For VM selection, VM is first selected from the VMs with assigned tasks, which can make the scheduling more compact. Let $\hat{t}^*$ represent the maximum finish time of all scheduled tasks. One of the characteristics of workflow tasks is parallelism, and $\hat{t}^*$ is often the dividing point of task execution in two adjacent levels. Therefore, $\hat{t}^*$ and the VMs with running tasks at the two adjacent levels are used as compact scheduling conditions.

Available start time refers to the earliest start time when a task is assumed to be executed on a specified VM, which is determined by the actual finish time of its predecessor task and the current completion time of the VM. For example, $\tilde{t}_{ih}^A \in \tilde{T}^A$ represents the available start time of task a_i on VM v_h. It is obtained as given in Eq. (23).

$$\tilde{t}_{ih}^A = \begin{cases} \bar{t}_h^*, & if\ Pre(a_i) = \varnothing, \\ \max\left\{ \max\limits_{a_j \in Pre(a_i)} \left\{ t_j^F + t_{ji}^{kh} \right\}, \bar{t}_h^* \right\}, & otherwise, \end{cases} \tag{23}$$

where $k = v_j^A$ is the assigned VM of task a_j. $\bar{t}_h^*$ is the current completion time of VM v_h and not less than the duration of cold startup if VM v_h is new created.

The start time of a task a_i on all available VMs can be obtained by Eq. (23). We need to select a VM to execute the task and determine the actual start time of the task (i.e. t_i^S). For compact scheduling, we prefer to assign the task with that at the same topology level together. In order to reduce data transmission among VMs, we prefer to assign the task with its predecessors together. Therefore, VMs that can execute this task are regarded as candidate set and divided into the following three layers:

Algorithm 1: T2FA

Input: Resource (P, U, B, M), workflow (A, W, E, D), $deadline$
Output: $\Pi = (V^A, T^S, T^F)$

1 Simplify DAG by Eq. (16);
2 Compute A^L using Eqs. (13) and (14);
3 $V \leftarrow \varnothing,\ V^P \leftarrow \varnothing$;
4 $k \leftarrow \arg\max\{U(h)|h \in P\}$;
5 $V^C \leftarrow \{k\}, \hat{t}^* \leftarrow \frac{\max\{w_i|a_i \in a_0^L\}}{U(k)}$;
6 **for** $l \leftarrow 0$ **to** $\max\{Lev\}$ **do**
7 **if** $(|a_l^L| = 1)$ *and* $(\hat{t}^E_{a_{i1}^L} > 0.1 \times \max\{\tilde{t}_i^F | a_i \in A\})$ **then**
8 Deploy the task to VM k that can be finished at the earliest;
9 **if** $k \notin V$ **then** $V \leftarrow V \cup \{k\}$;
10 $V^P \leftarrow \{k\}\ ,\ V^C \leftarrow \varnothing$;
11 $Continue$;
12 **end**
13 Randomly generate a rank from 1 to 4, denoted by $Rank$;
14 **foreach** $r \in Rank$ **do**
15 $\pi \leftarrow a_l^L \cap Type_r$;
16 $Type_r \leftarrow Type_r - \pi$;
17 $a_l^L \leftarrow a_l^L - \pi$;
18 Sort the tasks in π in descending order of weight value;
19 call $TaskSchedule(\pi)$;
20 **end**
21 Sort the unscheduled tasks in a_l^L in descending order of weight value;
22 call $TaskSchedule(a_l^L)$;
23 $V^P \leftarrow V^C,\ V^C \leftarrow \varnothing$;
24 **end**

- The first layer is the VM set V^C with running tasks at current topological level;
- The second layer is the VM set V^P with running tasks at immediately preceding topological level;
- The third layer is all available VMs, including leased VMs set V and non-leased VMs set with all VM types.

The VM candidate sets are traversed layer by layer until $\tilde{T}^A$ is determined. In each layer, if the smallest $\tilde{T}^A$ is less than $\hat{t}^*$, the corresponding $\tilde{T}^A$ is taken as the t_i^S.

Through the above process, the actual start time t_i^S of the task a_i and the VM v_i^A deployed by the task can be obtained simultaneously. The actual finish time of task a_i (i.e. t_i^F) is equal to the sum of actual start time and execution time. See Algorithm 2 for details.

4.1.4 Procedure for T2FA

T2FA is designed according to the characteristics of resource model and workflow application model, the detail of which are given in Algorithm 1.

Pre-processing (lines 1-5 in Algorithm 1). Through the structural decomposition of DAG in Section 4.1.2, DAG can be simplified by Eq. (16) (line 1 in Algorithm 1). Then divide the task topological level and determine the tasks at each level (line 2 in Algorithm 1). Set the candidate VMs set and the expected maximum finish time $\hat{t}^*$ (lines 3-5 in Algorithm 1). For DAG-based workflow scheduling, the selection of the first VM is crucial, which lays the foundation of scheduling.

Algorithm 2: TaskSchedule(π)

Input: Tasks order π
Output: Scheduling result of these tasks

1 **foreach** $i \in \pi$ **do**
2 **if** $V^C \neq \varnothing$ **then** $k \leftarrow \arg\min\{\tilde{t}_{ih}^A | h \in V^C\}$;
3 **if** $(V^C = \varnothing)$ *or* $(\tilde{t}_{ik}^A + t_{ik}^E > \hat{t}^*)$ **then**
4 **if** $V^P \neq \varnothing$ **then** $k \leftarrow \arg\min\{\tilde{t}_{ih}^A | h \in V^P\}$;
5 **if** $(V^P = \varnothing)$ *or* $(\tilde{t}_{ik}^A + t_{ik}^E > \hat{t}^*)$ **then**
6 $k \leftarrow \arg\min\{\tilde{t}_{ih}^A | h \in V \cup P\}$;
7 **if** $k \notin V$ **then** $V \leftarrow V \cup \{k\}$;
8 **end**
9 **end**
10 $v_i^A \leftarrow k, t_i^S \leftarrow \tilde{t}_{ik}^A, t_i^F \leftarrow t_i^S + t_{ik}^E$;
11 **if** $t_i^F > \hat{t}^*$ **then** $\hat{t}^* \leftarrow t_i^F$;
12 **if** $k \notin V^C$ **then** $V^C \leftarrow V^C \cup \{k\}$;
13 **end**

Task scheduling (lines 6-24 in Algorithm 1). In the task scheduling stage, it is divided into four levels according to the topology level and task type.

- Level 1 is the topological level from low to high (line 6 in Algorithm 1);
- Level 2 is the tasks of $Type_0$ (lines 7-12 in Algorithm 1);
- Level 3 is the tasks the other four special types (lines 13-20 in Algorithm 1);
- Level 4 is the tasks of general type (lines 21-22 in Algorithm 1).

The four special types $Type_1$-$Type_4$ are scheduled in random order. For tasks of same type, they are arranged in descending order of task weight, as scheduled in Algorithm 2. Algorithm 2 gives the details to get the actual start time of tasks and allocating VMs. ***In particular*** (line 8 in Algorithm 1) and lines 2, 4 and 6 in Algorithm 2), when there are multiple equal earliest $\tilde{T}^A$, the task is deployed to the VM instance with the shortest delay for the completion time.

Expected maximum finish time $\hat{t}^*$ is used as a reference value, and its initial value is set in line 5 of Algorithm 1. When scheduling tasks, VM with running tasks is preferentially selected. When the available finish time of task is less than $\hat{t}^*$, deploy the task to VM with the earliest available finish time; otherwise, select the VM with earliest available finish time from all VMs and update $\hat{t}^*$. The purpose of setting $\hat{t}^*$ is to make scheduling more compact. When there is no VM in leased VMs set V meeting the condition (less than $\hat{t}^*$), a new VM will be applied from the cloud platform and added to leased VMs set V (line 7 in Algorithm 2).

4.2 Delay Operation Based on Block Structure (DOBS)

When designing scheduling algorithm, there is generally a commonality: the task is executed as early as possible. If subsequent tasks cannot immediately start executing in time due to dependency constraints, a certain amount of idle time will be generated. However, part of the idle time can be avoided by delaying the start execution time of some tasks.

Definition 1 (Block Structure)**.** *It consists of tasks that are continuously executed without idle intervals on the same VM.*

Particularly, when there is only one task, it can also be called a block structure.

For example, the block structures in Fig. 4(a) are as follows: $[a_1, a_2, a_4, a_7]$, $[a_{5\_8}, a_3]$, $[a_6]$, $[a_9]$.

Theorem 1 (Block Structure Property). *In a given scheduling solution, first block structure on VM v_h is $X = [1, 2, ..., |X|]$. When $\forall x \in X, \hat{t}_x^F > t_x^F$, delaying the start time of the first block structure on the VM can reduce idle time and cost.*

Proof. Let $|X|+1$ denote the immediate succession task of first block structure. The idle time behind the block structure is the difference between the actual start time of task $|X|+1$ and the actual finish time of task $|X|$, that is, $t_{|X|+1}^S - t_{|X|}^F$.

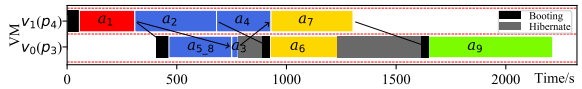
Let $\hat{t}_x^F$ denote the estimated latest finish time of task x in the current solution, which is the minimum difference between the start time and the transmission time of all direct successor tasks $Suc(x)$, except the successor tasks in the block structure. When $Suc(x) = \varnothing$, let $\hat{t}_x^F = t_x^F$. Thus,

$$\hat{t}_x^F = \begin{cases} \min\left\{t_y^S - C_{x,y}^{v_x^A, v_y^A}\right\}, y \in \left(Suc(x) - Suc(x) \cap X\right), \\ t_x^F, \qquad\qquad if\ Suc(x) = \varnothing. \end{cases} \tag{24}$$

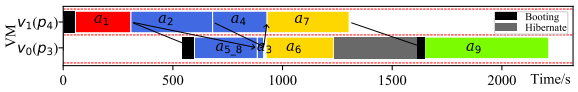
When $\forall x \in X, \hat{t}_x^F - t_x^F > 0$, the block structure can be moved backward by Δt without affecting the execution of other tasks.

$$\Delta t = \min\left\{t_{|X|+1}^S - t_{|X|}^F, \min\left\{\hat{t}_x^F - t_x^F | x \in X\right\}\right\}. \tag{25}$$

Therefore, when the start execution time of block structure is delayed by Δt, the cost saved is at least $M^H \times \lfloor \Delta t \rfloor$. At the same time, due to the reduction of idle time, the resource utilization of VM will inevitably increase. □



(a) Before delay operation.



(b) After delay operation.

Fig. 4. Comparison before and after delay operation based on block structure.

As shown in Fig. 4, when tasks a_{5_8} and a_3 satisfy Theorem 1, their start time can be delayed by $\Delta t = \min\left\{t_{a_6}^S - t_{a_3}^F, \min\left\{\hat{t}_{a_{5\_8}}^F - t_{a_{5\_8}}^F, \hat{t}_{a_3}^F - t_{a_3}^F\right\}\right\} = \min\left\{146.1, \min\left\{897.1, 134.3\right\}\right\} = 134.3$. The process of delaying tasks can also take advantage of the pay-as-you-go feature of cloud to save cost. Therefore, Theorem 1 is applied to adjust the scheduling solution of T2FA. Traverse the first block structure of each VM until there is no block structure that satisfies the constraint of Theorem 1. If the block structure satisfies the constraint of Theorem 1, the actual start and finish time of related tasks will be updated. See the Algorithm 3 for details.

4.3 Instance Hibernate Scheduling Heuristic (IHSH)

Recently, cloud service providers have provided some instances that support hibernation function. If one instance is kept being idle for a period, it is wise to hibernate it to save cost. Nevertheless, frequent hibernation may lead to system failure or software operation error. Hence, a heuristic is proposed to schedule when to hibernate an instance. Once each state is determined, the total cost and total idle rate can be obtained. Traverse each idle interval between in instance, and set the idle interval to hibernation mode when the requirement of hibernation is met. The ***requirements of hibernation*** may be the shortest duration of hibernation and the minimum gap between two adjacent hibernation in one instance. According to the scheduling results of T2FA and DOBS, the tasks executed on each instance and their start and finish time are known. Let S_{ij} represent the jth task executed on instance v_i. See the Algorithm 4 for details.

Algorithm 3: DOBS

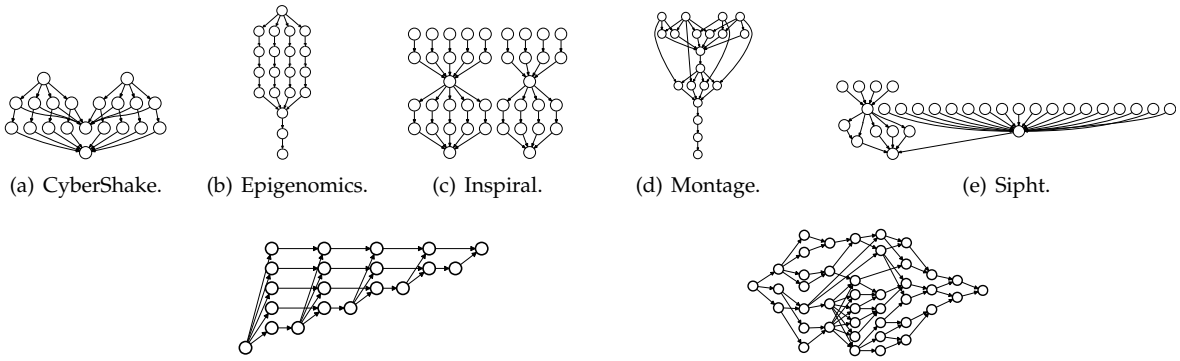
Input: $\Pi = (V^A, T^S, T^F)$
Output: New $\Pi = (V^A, T^S, T^F)$
repeat
  foreach $h \in V$ **do**
    Find X as first block structure in instance v_h;
    Compute $\hat{t}_x^F$ using Eq. (24), $\forall x \in X$;
    if $\forall x \in X, \quad \hat{t}_x^F > t_x^F$ **then**
      Compute Δt using Eq. (25);
      foreach $x \in X$ **do**
        $t_x^S \leftarrow t_x^S + \Delta t$;
        $t_x^F \leftarrow t_x^F + \Delta t$;
      end
    end
  end
until *no block is found*;

Algorithm 4: IHSH

Input: $\Pi = (V^A, T^S, T^F)$
Output: $R = (\bar{T}^S, \bar{T}^E, \bar{T}^{HS}, \bar{T}^{HE})$,
$f(\Pi, R) = (total\ cost, total\ idle\ rate)$
foreach $h \in V$ **do**
  $tempT \leftarrow 0, j \leftarrow 1$;
  for $k \leftarrow 1$ **to** $|S_h| - 1$ **do**
    $p \leftarrow S_{hk}, s \leftarrow S_{h(k+1)}$;
    if $t_s^S - t_p^F > Dur^H$ & $t_p^F - tempT > Gap^H$ **then**
      $\bar{t}_{hj}^{HS} \leftarrow t_p^F, \bar{t}_{hj}^{HE} \leftarrow t_s^S - Dur^W$;
      $tempT \leftarrow t_s^S, j \leftarrow j + 1$;
    end
  end
  $p \leftarrow S_{h1}, s \leftarrow S_{h|S_h|}$;
  $\bar{t}_h^S \leftarrow t_p^S - Dur^P, \bar{t}_h^E \leftarrow t_s^F$;
end
Compute $f(\Pi, R) = (total\ cost, total\ idle\ rate)$.

4.4 Time Complexity of ET2FA

The time complexity of ET2FA depends on its three phases. Let n be the number of tasks and $e$be the number of edges in workflow. Since the maximum number of edges is $\frac{(n-1)(n-2)}{2}$ in DAG, assume $e \simeq O(n^2)$. Let $|V|$ be the maximum number of VMs required. In fact, the maximum number of VMs required will not exceed n, and Wu et al. [11] have proved



(a) CyberShake. (b) Epigenomics. (c) Inspiral. (d) Montage. (e) Sipht.

(f) Gaussian Elimination of problem size 6 ($n = 20$). (g) Molecular Dynamics Code.

Fig. 5. Seven real-world workflow applications.

that $|V| \leq n - \max\{Lev\}$, so assume $|V| \simeq O(n)$. The time complexity of the ET2FA is analyzed as follows:

- *T2FA*: $O(n^2)$. In lines 1-2 in Algorithm 1, simplify DAG and compute A^L must traverse all tasks, which can be done within $O(n)$ and $O(n+e)$ respectively. In lines 6-24 in Algorithm 1, it is essentially to traverse each task and allocate resources for the task through $\tilde{T}^A$. In the process of traversing each task, tasks are divided into different types. In the worst case, there is only one type (only lines 21-22 are executed). The time complexity of sorting is $O(n^2)$ in line 21. In line 2 in Algorithm 2, the time complexity of $\tilde{T}^A$ is $O(|V|)$. The time complexity of Algorithm 2 is $O(n\,|V|)$. Therefore, the time complexity of lines 6-24 in Algorithm 1 is $O(n \log n)+O(n\,|V|)$. In summary, the time complexity of T2FA is $O(n)+O(n+e)+O(n^2)+O(n\,|V|)=O(n^2)$.
- *DOBS*: $O(n^2)$. In the worst case, there is only one block structure on each VM, that is, each task needs to be traversed. Eqs. (24)-(25), the time complexity of LFT and Δt are $O(n+e)$, and $O(n+|V|)$, respectively. Therefore, the time complexity of DOBS is $O(n+e)O(n+|V|) = O(n^2)$.
- *IHSH*: $O(n)$. IHSH needs to traverse each task with a time complexity of $O(n)$.

According to the above analysis, the time complexity of ET2FA is $O(n^2) + O(n^2) + O(n) = O(n^2)$.

5 Performance Evaluation

5.1 Simulation Environment

5.1.1 Resource Environment

In simulation experiment, we use 5 representative VM types from low configuration to high configuration. The VM configurations and their processing capacity are based on current Amazon EC2 platform , as presented in Table 4. According to the researches in [3], [4], [36], the processing capacity in GFLOPS is estimated based on the number of EC2 compute units (ECU). One ECU currently provides CPU capacity equivalent to a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor. EC2 usage are billed on one second increments, with a minimum of 60 seconds. As for the booting time of VM, according to the researches of [37], [38], the cold startup time has been reduced from 97s to 55.9s for Amazon EC2 Cloud. The warm startup time is set to 34.0s based on the results obtained by [38]. Other parameters and their values are listed in Table 5.

Table 4
Configurations and Prices of Virtual Machines[2].

	VM Type	ECU	Processing Capacity (GFLOPS)	Cost ($/h)	Bandwidth (Gbps) [a]
1	c3.large	7	30.8	0.128	1
2	c3.xlarge	14	61.6	0.255	1.5
3	c3.2xlarge	28	123.2	0.511	2
4	c3.4xlarge	55	242	1.021	3
5	c3.8xlarge	108	475.2	2.043	3

a. Manually set according to the VM configuration.

Table 5
List of Other Parameters and Their Values.

Parameter	Symbol	Value
The duration of cold startup	Dur^C	55.9s
The duration of warm startup	Dur^W	34.0s
The duration of stopping	Dur^P	5.6s
The shortest duration of hibernation	Dur^H	60.0s
The minimum gap between two adjacent hibernation in one instance	Gap^H	120s
ElasticIP cost	M^H	0.005$/h

5.1.2 Workflow Applications

Seven real-world workflow applications with different scales (numbers of tasks) from different scientific areas are adopted in the simulation, as shown in Fig. 5.

The following five real-world workflow applications have benchmark data: CyberShake, Epigenomics, Inspiral, Montage and Sipht [4], [12], [13], [16], [39]. Figs. 5(a)-5(e) show the sample DAG structures of these workflows [1], [34]. More details about these workflows can be found in [39]. All these workflows are generated in form of Directed Acyclic Graph in XML (DAX) format by Pegasus Workflow-Generator [13], [39], and are publicly available on Pegasus website[3]. These DAX files contain information such as list of tasks, dependencies between tasks, their computation time and size of the input/output files generated by the tasks. Similar to [40], these benchmarks are adaptively adjusted. That is, it is assumed that these benchmarks are simulated and generated on a processor with the same configuration as VM p_3. The number of tasks varies from 24-1000.

2. https://aws.amazon.com/ec2/pricing/on-demand/, https://instances.vantage.sh/

3. https://confluence.pegasus.isi.edu/display/pegasus/Deprecated+Workflow+Generator

The following two real-world workflow applications only have DAG structure: Gaussian elimination and Molecular dynamics code [7]. In the experiment, the weight of tasks W is randomly generated from a uniform distribution $[1800, 180000]$ and the amount of data transmitted between tasks D is randomly generated from $[18, 1800]$. The Gaussian elimination is an algorithm used to solve a system of linear equations. The total number of tasks n in a Gaussian elimination graph is determined by the matrix size m, which is equal to $n = \frac{(m+2)(m-1)}{2}$ (see Fig. 5(f)). In the experiment, set $m = \{10, 20, 35, 45\}$ and the corresponding $n = \{54, 209, 629, 1034\}$. The molecular dynamics code is given in [7], as shown in Fig. 5(g). This application has a fixed DAG structure and the number of tasks $n = 41$.

To distinguish different problems, a symbol 'Workflow type_Number of tasks' is adopt , such as 'Cyber_30' represents CyberShake of 30 tasks. In particular, 'Molec_0' represents the 0th test problem of Molecular dynamics code.

These seven workflow applications have different structures and characteristics and are widely used to evaluate the performance of the workflow scheduling approaches. Their specific characteristics are listed in Table 6. They can be decomposed into at least two structures in Fig. 3, and all of them have SOMI structure (Fig. 3(c)).

Table 6
The Structures in Fig. 3 Contained in Different Workflows.

Workflow	SOSI	MOSI	SOMI	MOMI
CyberShake		✓	✓	✓
Epigenomics	✓	✓	✓	
Inspiral	✓	✓	✓	
Montage	✓		✓	
Sipht			✓	✓
Gaussian elimination	✓	✓	✓	✓
Molecular dynamics code		✓	✓	✓

5.2 Baseline Algorithms

To illustrate the effectiveness of the proposed algorithm, five baseline algorithms are implemented for comparison, including two heuristic algorithms IC-PCP [12] and JIT-C [13], two meta-heuristic algorithms PSO [4] and KADWWO [22], and a reinforcement learning algorithm QL-HEFT [16]. These five algorithms are classical in solving the cost-minimization and the deadline-constrained workflow scheduling problem. As mentioned in Section 2, IC-PCP divides partial critical paths based on deadline, and applies recursive method to schedule tasks to optimize cost. JIT-C schedules tasks sequentially based on task topological level, combines the cheapest task-VM mapping for VM selection, and optimizes cost. PSO adopts the sequence based task to resource mapping method (encoding), schedules tasks to specified VM (decoding) without violating the dependency between tasks, and then integrates the coding and decoding into PSO's iterative mechanism to optimize cost. KADWWO adopts the coding and decoding schemes similar to PSO, and designs the discrete propagation operator, adaptive refraction operator and breaking operator of WWO. The optimization objective is to minimize cost under deadline constraint. QL-HEFT regards tasks as states and actions respectively, takes the rank value in HEFT as immediate reward, obtains the scheduling order of tasks through Q table, and then selects VM by the earliest finish task rule. It is evaluated on several metrics such as makespan, efficiency and average response time.

The time complexity of IC-PCP and JIT-C is equal to that of ET2FA, which is $O(n^2)$. The time complexity of PSO and KADWWO is $O(pgn^2)$, where p is the population size and g is evolutionary generations. The time complexity of QL-HEFT is $O(gn^2)$, where g is the number of iterations. In QL-HEFT, g is different from that of PSO, which is not a definite value and is limited by convergence conditions and running time. It can be seen that the time complexity of ET2FA is the same as that of other heuristic algorithms, which are smaller than PSO, KADWWO and QL-HEFT. This is also proved by the comparison of running time of algorithms.

The parameters of PSO are set according to the optimal parameters given in [4], which are $c_1 = c_2 = 2.0$, $\omega = 0.5$, and the number of particles and iteration times are set to 100. The parameters of KADWWO are set according to the optimal parameters given in [22], which are $NP = 5$, $c = 0.4$, $M = 0.5$, $q = 0.3$, and the maximum number of fitness evaluation is set to $n \times 100$. For different problems, the running time of each algorithm does not exceed $1.2 \times n$ seconds. Another termination condition of QL-HEFT is that the target value does not change for ten consecutive times, and it is considered that the algorithm converges.

Since this paper is the first time to study instance hibernation to save cost, for the sake of fairness, the third stage IHSH is applied to all baseline algorithms. Each algorithm is repeated ten times, and the mean value is taken as the final solution obtained by the algorithm. All algorithms are coded in Python and are executed on Intel Core i5-9500 3.0GHz processor with 32GB RAM.

5.3 Performance Results

To evaluate the impacts of different workflow types and resource quantity, for each workflow under different deadline factors, comparisons of the algorithms are based on the following three metrics: total cost, total idle rate and running time of the algorithms. Since workflows with different types and different scales of task numbers have different scales of costs, total cost and total idle rate should be normalized before they can be aggregated for comparison. The original data of the experimental results are available at https://github.com/szx1010/ET2FA-Performance-Results. Therefore, the Relative Percentage Deviation (RPD) is used as the response variable for evaluating the results [24], [41] and it is defined as follows:

$$RPD^A = \frac{f^A - f_{min}}{f_{max} - f_{min}}, \tag{26}$$

where f^A is the solution obtained by algorithm A, and f_{min} and f_{max} are the minimum and maximum value achieved among all the comparison algorithms, respectively. That is, the algorithm with $RPD = 0$ has the best effect on the minimization problem. Running time is the CPU execution time for the algorithm to obtain the scheduling solution for a given problem. Although the workflow scheduling problem is static scheduling problem, in order to provide a practical solution, running time is the key evaluation metric to measure the algorithm [1], [7], [35].

One way Analysis of Variance (ANOVA) technique is conducted for analysing the performance of the proposed

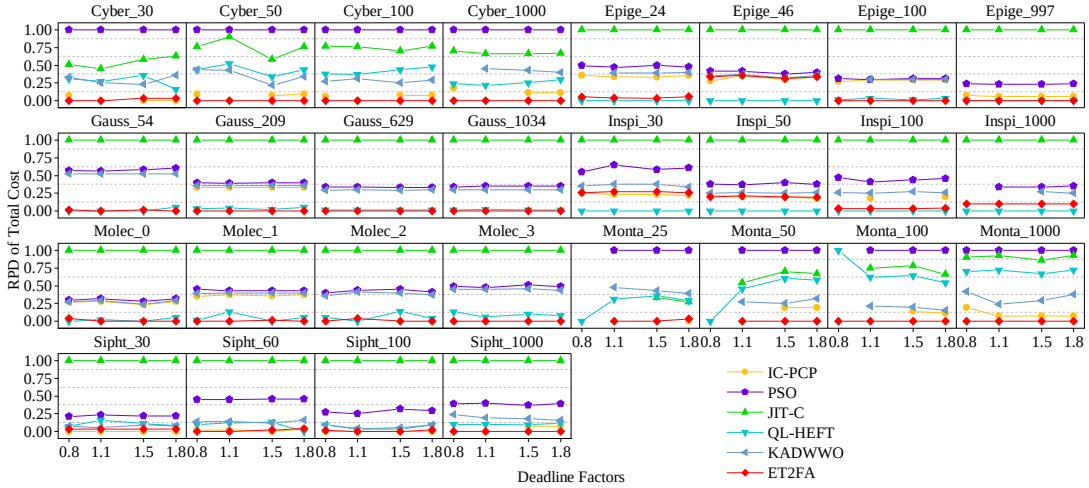


Fig. 6. The RPD of Total Cost of each workflow with IC-PCP, PSO, JIT-C, QL-HEFT, KADWWO, and ET2FA.

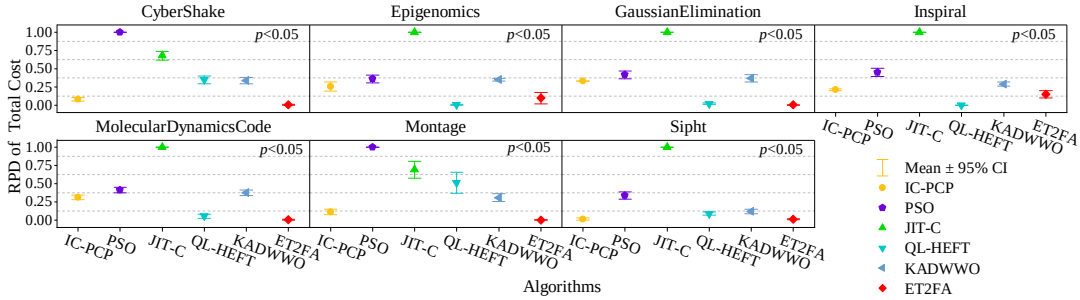
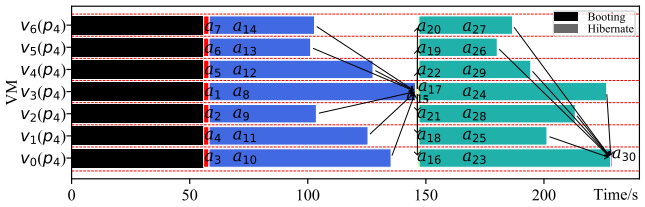


Fig. 7. Means plot of RPD of Total Cost with 95.0 percent Tukey HSD confidence intervals.

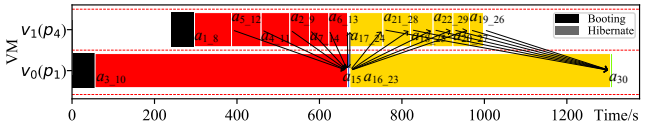
algorithm whether there are statistically significant differences in the results. The significance level is set to 0.05. As we know, the ANOVA is frequently employed in the literature due to an effective statistical analysis [41], [42]. When the solution obtained by the algorithm cannot satisfy the deadline constraint (i.e. infeasible solution), the values of the three metrics are null.

5.3.1 Comparison of Total Cost

Fig. 6 shows the RPD of Total Cost of each workflow with different algorithms. For CyberShake workflow, ET2FA and IC-PCP are obviously superior to other algorithms, but when deadline factor is 1.1, IC-PCP can't get a feasible solution. For Epigenomics, Gaussian elimination and Molecular dynamics code workflows, except for 'Epige_46', ET2FA has almost the same performance as QL-HEFT, which is superior to other algorithms. Among them, IC-PCP can't get feasible solutions for Gaussian elimination workflows except 'Gauss_209'. For Inspiral workflow, QL-HEFT is the best, followed by ET2FA. For Montage workflow, when the deadline factor is 0.8, only QL-HEFT can get a feasible solution on 'Monta_25' and 'Monta_50'. For other problems of Inspiral workflow, ET2FA can achieve better performance. For Sipht workflow, the performance of IC-PCP, QL-HEFT and ET2FA is not significantly different, especially for IC-PCP and ET2FA, whose performances are almost the same. In addition, these three algorithms outperform PSO, KADWWO and JIT-C. For all of Epige_24, Epige_46 and Inspiral workflows, ET2FA is slightly worse than QL-HEFT due to the same reason. Fig. 8 is an example showing the comparison results under workflow Inspi_30. As shown



(a) Gantt chart of scheduling with Inspi_30 over QL-HEFT, ($total\ cost = 0.32, total\ idle\ rate = 2.85$).



(b) Gantt chart of scheduling with Inspi_30 over ET2FA, ($total\ cost = 0.52, total\ idle\ rate = 0.12$).

Fig. 8. Comparison Inspiral workflow scheduling of 30 tasks over QL-HEFT and ET2FA, ($deadline = 2136$).

in Fig. 8(a), QL-HEFT evenly distributes tasks to multiple VMs with high performance. As shown in Fig. 8(b), ET2FA assigns task a_{3_10} to the VM type with low performance, resulting in a longer completion time and higher cost. But this still meets the deadline. The fundamental reason is that when there is a tie (i.e. multiple equal available start time $\tilde{T}^A$) in line 8 of Algorithm 1 and lines 2, 4 and 6 of Algorithm 2, the VM type with low performance is selected.

Fig. 7 is plot of RPD of Total Cost with 95.0 percent Tukey HSD confidence intervals of all algorithms for all workflows. All p-values are less than 0.05, indicating that all algorithms have a significant different on the response variable at the 95.0% confidence level.

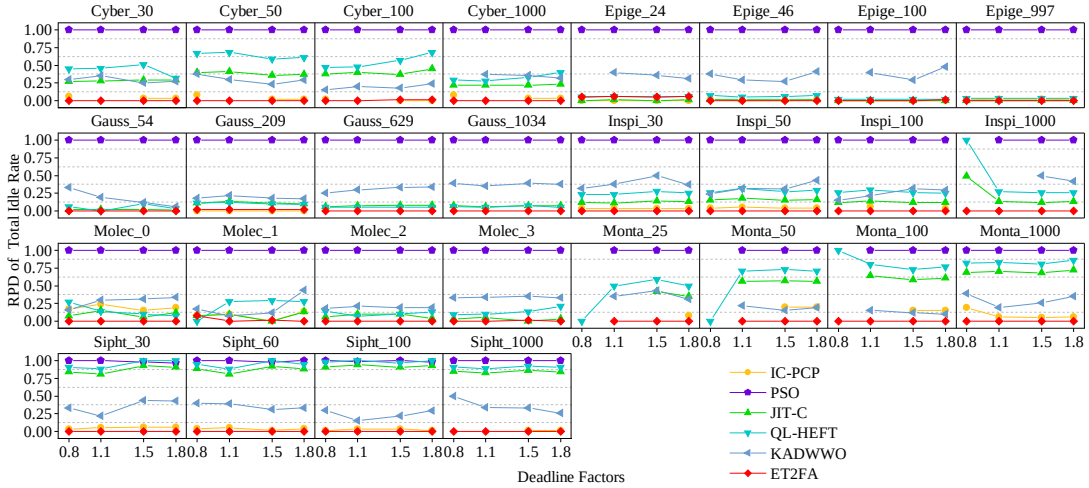


Fig. 9. The RPD of Total Idle Rate of each workflow with IC-PCP, PSO, JIT-C, QL-HEFT, KADWWO, and ET2FA.

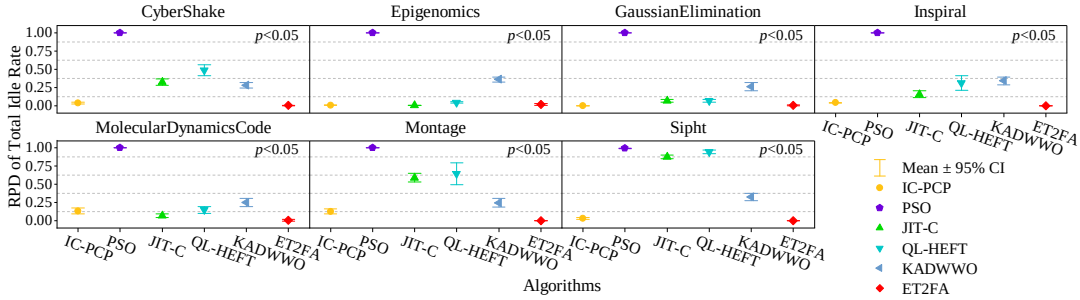


Fig. 10. Means plot of RPD of Total Idle Rate with 95.0 percent Tukey HSD confidence intervals.

The better performance of IC-PCP is mainly owing to its critical path method: when allocating resources for one critical path, choose from low allocation until there is a VM that can deploy the whole path. This method basically does not cause additional data transmission cost. The main reasons why PSO, KADWWO and JIT-C algorithm can't always achieve better performance are as follows: (1) PSO and KADWWO are swarm intelligence optimization algorithms, which are random in initialization and need constant iterative optimization to achieve better performance; When selecting resources for tasks, these two algorithms randomly select resources without guidance, and do not consider the impact of data transmission. (2) JIT-C has the cheapest selection rule when allocating resources for tasks. However, it may be the cheapest to select resources at a certain time, but not necessarily the cheapest for the whole workflow.

5.3.2 Comparison of Total Idle Rate

Fig. 9 shows the RPD of Total Idle Rate of each workflow with different algorithms. Fig. 10 shows means plot of RPD of Total Idle Rate with 95.0 percent Tukey HSD confidence intervals of all algorithms for all workflows. All p-values are less than 0.05 which indicates that all algorithms have a significant different on the response variable at the 95.0% confidence level. Total idle rate mainly evaluates the full utilization of the selected VM by the algorithm. As can be seen from Fig. 10, for Epigenomics, Gaussian elimination and Molecular dynamics code workflows, the performance of PSO is worse than other algorithms, and the performance of other algorithms is almost the same with no significant difference. For CyberShake, Inspiral and Sipht workflows, IC-PCP and ET2FA are superior to other algorithms, and their performance is almost the same, with no significant difference. For Montage workflow, ET2FA is superior to other algorithms and has significant differences.

The performance of PSO is always the worst, which shows that the algorithm can't make full use of the selected VM. That is, when allocating resources for tasks, it is necessary to dynamically allocate resources in the scheduling process, rather than selecting fixed resource types for tasks in advance. It can be seen from Fig. 9 that QL-HEFT is always close to PSO, and its performance is also poor. The reason is that when allocating resources, QL-HEFT always selects resources according to the rule of the earliest completion time. This rule may turn this problem into a homogeneous resource type with only one highest configured VM type. Although QL-HEFT can't achieve good performance for total idle rate, it can perform well for total cost, especially for Epigenomics and Inspiral workflows.

5.3.3 Running Time of the Algorithms

Table 7 shows the average running time of the algorithm under different workflow types. Avg is the average running time of the algorithm based on all workflow types. The time of ET2FA is always the least, and its average time is only 0.346 seconds. With the increase of the scale of the problem, the running time does not change significantly. Since PSO and KADWWO are swarm intelligence algorithms, through iterative optimization, they are obvious that the running time of the algorithm is longer than that of the heuristic algorithm. QL-HEFT is also an iterative optimization algorithm by constantly updating Q-table. Although IC-PCP and JIT-C are heuristic algorithms, there are still some iterations

in the algorithm process. As the problem scale increases, the running time becomes significantly longer.

Table 7
Average Running Time of Scheduling Algorithms (in Second).

	IC-PCP	PSO	JIT-C	QL-HEFT	KADWWO	ET2FA
Cyber_30	0.054	36.498	0.380	0.655	26.653	**0.034**
Cyber_50	0.085	61.485	0.998	0.662	61.877	**0.059**
Cyber_100	0.314	123.247	4.121	2.100	127.071	**0.128**
Cyber_1000	16.305	1219.139	389.530	199.216	1458.821	**1.510**
Epige_24	0.033	29.238	0.041	0.257	11.698	**0.014**
Epige_46	0.063	57.286	0.122	2.528	42.893	**0.026**
Epige_100	0.373	121.802	0.344	2.123	124.906	**0.054**
Epige_997	8.886	1218.974	25.965	436.005	*null*	**0.562**
Gauss_54	*null*	65.850	0.863	53.767	53.929	**0.058**
Gauss_209	0.821	254.838	13.483	251.488	255.339	**0.262**
Gauss_629	*null*	768.435	150.028	760.295	789.016	**0.758**
Gauss_1034	*null*	1268.986	365.238	1250.637	1435.058	**1.314**
Inspi_30	0.039	36.503	0.119	0.231	19.182	**0.021**
Inspi_50	0.067	60.837	0.272	0.600	51.858	**0.033**
Inspi_100	0.168	121.688	0.999	3.992	124.277	**0.068**
Inspir_1000	*null*	1220.206	84.327	167.148	1432.261	**0.829**
Molec_0	0.058	50.075	0.660	23.715	31.550	**0.046**
Molec_1	0.056	50.068	0.660	7.757	31.830	**0.046**
Molec_2	*null*	49.891	0.660	16.180	31.992	**0.047**
Molec_3	*null*	49.939	0.659	11.955	31.723	**0.046**
Monta_25	0.030	30.290	0.222	0.217	21.012	**0.030**
Monta_50	0.086	61.307	0.987	0.885	62.503	**0.069**
Monta_100	0.281	123.058	4.038	2.095	127.815	**0.139**
Monta_1000	73.626	1246.531	469.442	187.338	1942.837	**2.312**
Sipht_30	0.045	35.128	0.296	0.583	35.879	**0.028**
Sipht_60	0.090	71.047	1.067	0.857	73.640	**0.055**
Sipht_100	0.167	118.190	2.845	1.918	129.099	**0.090**
Sipht_1000	13.476	1184.152	277.391	161.450	1811.342	**1.064**
Avg	5.233	347.667	64.134	126.666	383.187	**0.346**

When these results are combined, the ET2FA is an effective and efficient algorithm for workflow scheduling. The superiority of ET2FA are as follows: 1) ET2FA considers three special structures in DAG and prioritizes the tasks of these structures; 2) We devises a guiding VM selection method based on compact scheduling conditions; 3) We further optimizes the scheduling results by utilizing the property of block structure.

6 Conclusion

In this paper, a more realistic workflow scheduling problem in cloud with hibernation mode and per-second billing with a minimum of 60 seconds is considered. By analyzing the characteristics of the problem and the properties of the block structure, a hybrid heuristic algorithm with three stages is proposed, which is called Enhanced Task Type Priority Algorithm (ET2FA). The simulation results and comparisons demonstrate that ET2FA outperforms the baseline algorithms including two heuristic algorithms IC-PCP and JIT-C, two meta-heuristic algorithms PSO and KADWWO, and a reinforcement learning algorithm QL-HEFT.

ET2FA outperforms baselines, but it still has the following limitations: 1) We ignore the heterogeneous tasks such as computing-intensive, memory-intensive, network-intensive and GPU-demanding tasks; 2) When the scale of workflow becomes very large, ET2FA may not guarantee a satisfactory solution due to its small search space.

In the future, we intend to further enhance the quality of the presented method on large/very large-scale workflow scheduling with energy optimization (e.g., up to 5000 tasks). In the case of a large-scale tasks, heuristic algorithm is limited to a small search space, and its solution can still be further optimized, so we intend to integrate heuristic algorithm into meta-heuristic algorithm to obtain the non-dominated solution set.

Acknowledgments

This work is financially supported by Shenzhen Science and Technology Program under Grant No.JCYJ20210324132406016, National Natural Science Foundation of China under Grant No.61732022 and Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies under Grant No.2022B1212010005.

References

[1] A. Song, W.-N. Chen, X. Luo, Z.-H. Zhan, and J. Zhang, "Scheduling workflows with composite tasks: A nested particle swarm optimization approach," *IEEE Trans. Serv. Comput.*, vol. 15, no. 2, pp. 1074–1088, 2020.

[2] H. Yuan, J. Bi, and M. C. Zhou, "Energy-efficient and QoS-optimized adaptive task scheduling and management in clouds," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 2, pp. 1233–1244, 2022.

[3] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, and B. Luo, "Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds," *IEEE Trans. Serv. Comput.*, vol. 11, no. 4, pp. 713–726, 2018.

[4] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, 2014.

[5] Q. Z. Xiao, J. Zhong, L. Feng, L. Luo, and J. Lv, "A cooperative coevolution hyper-heuristic framework for workflow scheduling problem," *IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 150–163, 2022.

[6] Z. G. Chen, Z. H. Zhan, Y. Lin, Y. J. Gong, T. L. Gu, F. Zhao, H. Q. Yuan, X. Chen, Q. Li, and J. Zhang, "Multiobjective cloud workflow scheduling: a multiple populations ant colony system approach," *IEEE Trans. Cybern.*, vol. 49, no. 8, pp. 2912–2926, 2019.

[7] 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, 2002.

[8] J. Zhou, K. Cao, P. Cong, T. Wei, M. Chen, G. Zhang, J. Yan, and Y. Ma, "Reliability and temperature constrained task scheduling for makespan minimization on heterogeneous multi-core platforms," *J. Syst. Softw.*, vol. 133, pp. 1–16, 2017.

[9] C. G. Wu, W. Li, L. Wang, and A. Y. Zomaya, "Hybrid evolutionary scheduling for energy-efficient fog-enhanced internet of things," *IEEE Trans. Cloud Comput.*, vol. 9, no. 2, pp. 641–653, 2021.

[10] Z. H. Zhan, X. F. Liu, Y. J. Gong, J. Zhang, H. S. H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surv.*, vol. 47, no. 4, 2015.

[11] H. Wu, X. Hua, Z. Li, and S. Ren, "Resource and instance hour minimization for deadline constrained DAG applications using computer clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 3, pp. 885–899, 2016.

[12] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Futur. Gener. Comp. Syst.*, vol. 29, no. 1, pp. 158–169, 2013.

[13] J. Sahni and P. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 2–18, 2018.

[14] X. Li, L. Qian, and R. Ruiz, "Cloud workflow scheduling with deadlines and time slot availability," *IEEE Trans. Serv. Comput.*, vol. 11, no. 2, pp. 329–340, 2018.

[15] Y. H. Jia, W. N. Chen, H. Yuan, T. Gu, H. Zhang, Y. Gao, and J. Zhang, "An intelligent cloud workflow scheduling system with time estimation and adaptive ant colony optimization," *IEEE Trans. Syst. Man Cybern. -Syst.*, vol. 51, no. 1, pp. 634–649, 2021.

[16] Z. Tong, X. Deng, H. Chen, J. Mei, and H. Liu, "QL-HEFT: a novel machine learning scheduling scheme base on cloud computing environment," *Neural Comput. Appl.*, vol. 32, no. 10, pp. 5553–5570, 2020.

[17] P. K. Muhuri and S. K. Biswas, "Bayesian optimization algorithm for multi-objective scheduling of time and precedence constrained tasks in heterogeneous multiprocessor systems," *Appl. Soft. Comput.*, vol. 92, 2020.

[18] M. Adhikari, T. Amgoth, and S. N. Srirama, "A survey on scheduling strategies for workflows in cloud environment and emerging trends," *ACM Comput. Surv.*, vol. 52, no. 4, 2019.

[19] Z. Sun, C. Gu, H. Huang, and H. Zhang, "T2FA: A heuristic algorithm for deadline-constrained workflow scheduling in cloud with multicore resource," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, 2021, pp. 345–354.

[20] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Futur. Gener. Comp. Syst.*, vol. 27, no. 8, pp. 1011–1026, 2011.

[21] S. G. Domanal, R. M. R. Guddeti, and R. Buyya, "A hybrid Bio-inspired algorithm for scheduling and resource Management in cloud environment," *IEEE Trans. Serv. Comput.*, vol. 13, no. 1, pp. 3–15, 2020.

[22] S. Qin, D. Pi, Z. Shao, and Y. Xu, "A knowledge-based adaptive discrete water wave optimization for solving cloud workflow scheduling," *IEEE Trans. Cloud Comput.*, pp. 1–18, 2021.

[23] A. Deldari, M. Naghibzadeh, and S. Abrishami, "CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud," *J. Supercomputing*, vol. 73, no. 2, pp. 756–781, 2017.

[24] Z. Zhu and X. Tang, "Deadline-constrained workflow scheduling in IaaS clouds with multi-resource packing," *Futur. Gener. Comp. Syst.*, vol. 101, pp. 880–893, 2019.

[25] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, and S. Hu, "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT," *Futur. Gener. Comp. Syst.*, vol. 93, pp. 278–289, 2019.

[26] R. K. Naha and M. Othman, "Cost-aware service brokering and performance sentient load balancing algorithms in the cloud," *J. Netw. Comput. Appl.*, vol. 75, pp. 47–57, 2016.

[27] R. Sudarsan and C. J. Ribbens, "Combining performance and priority for scheduling resizable parallel applications," *J. Parallel Distrib. Comput.*, vol. 87, pp. 55–66, 2016.

[28] Z. J. Wang, Z. H. Zhan, W. J. Yu, Y. Lin, J. Zhang, T. L. Gu, and J. Zhang, "Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2715–2729, 2020.

[29] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia, and J. Wen, "Deadline-constrained cost optimization approaches for workflow scheduling in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3401–3412, 2017.

[30] M. S. Sanaj and P. M. Joe Prathap, "Nature inspired chaotic squirrel search algorithm (CSSA) for multi objective task scheduling in an IAAS cloud computing atmosphere," *Eng. Sci. Technol.*, vol. 23, no. 4, pp. 891–902, 2020.

[31] H. Y. Shishido, J. C. Estrella, C. F. M. Toledo, and M. S. Arantes, "Genetic-based algorithms applied to a workflow scheduling algorithm with security and deadline constraints in clouds," *Comput. Electr. Eng.*, vol. 69, pp. 378–394, 2018.

[32] D. Ding, X. Fan, Y. Zhao, K. Kang, Q. Yin, and J. Zeng, "Q-learning based dynamic task scheduling for energy-efficient cloud computing," *Futur. Gener. Comp. Syst.*, vol. 108, pp. 361–371, 2020.

[33] G. Ismayilov and H. R. Topcuoglu, "Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing," *Futur. Gener. Comp. Syst.*, vol. 102, pp. 307–322, 2020.

[34] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1344–1357, 2016.

[35] Q. Wu, M. C. Zhou, and J. Wen, "Endpoint communication contention-aware cloud workflow scheduling," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 2, pp. 1137–1150, 2022.

[36] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Pro. Int. Conf. Cloud Comput. CloudComp 2009*, 2010, pp. 115–131.

[38] J. Hao, T. Jiang, W. Wang, and I. K. Kim, "An empirical analysis of VM startup times in public IaaS clouds," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, 2021, pp. 398–403.

[37] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput. (CLOUD)*, 2012, pp. 423–430.

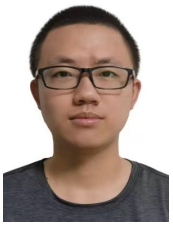
[39] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Futur. Gener. Comp. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.

[40] J. Zhou, J. Sun, M. Zhang, and Y. Ma, "Dependable scheduling for real-time workflows on cyber-physical cloud systems," *IEEE Trans. Ind. Inform.*, vol. 17, no. 11, pp. 7820–7829, 2021.

[41] L. Chen, X. Li, Y. Guo, and R. Ruiz, "Hybrid resource provisioning for cloud workflows with malleable and rigid tasks," *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 1089–1102, 2021.

[42] J. Wang, X. Li, R. Ruiz, J. Yang, and D. Chu, "Energy utilization task scheduling for MapReduce in heterogeneous clusters," *IEEE Trans. Serv. Comput.*, vol. 15, no. 2, pp. 931–944, 2022.

Zaixing Sun received the M.S. degree in Control Engineering from the Kunming University of Science and Technology, Kunming, China, in 2019. Currently, He is a Ph.D. candidate in Harbin Institute of Technology, Shenzhen, China. His research interests include cloud computing, intelligent optimization and scheduling.



Boyu Zhang is currently an undergraduate student in School of Artificial Intelligence, Changchun University of Science and Technology, Changchun, China. His research interests include algorithm design in cloud computing, privacy-preserving technology in cloud computing, etc.

Chonglin Gu received PhD degree in computer science and technology from Harbin Institute of Technology, Shenzhen in 2018. After that, he has been a postdoctoral fellow in the Chinese University of Hong Kong, Shenzhen, China. He is currently an assistant professor in the school of computer science and technology in Harbin Institute of Technology, Shenzhen. His research interests include cloud computing, especially algorithm design and system implementation.

Ruitao Xie received her PhD degree in Computer Science from City University of Hong Kong in 2014, and BEng degree from Beijing University of Posts and Telecommunications in 2008. She is currently an assistant professor in College of Computer Science and Software Engineering, Shenzhen University. Her research interests include edge computing, AI networking, cloud computing and distributed systems.

Bin Qian received the Ph.D. degree in control science and engineering from Tsinghua University, Beijing, China, in 2009. From 2018 to 2019, he was a visiting professor at Manchester Business School, The University of Manchester, Manchester, UK. He is currently a Professor with the School of Information Engineering and Automation, Kunming University of Science and Technology. His research interests include intelligent optimization and scheduling.

Hejiao Huang received her PhD degree in Computer Science from City University of Hong Kong in 2004. She is currently a professor in Harbin Institute of Technology, Shenzhen, China, and previously was an invited professor at INRIA, France. Her research interests include network security, cloud computing security, trustworthy computing, big data security, formal methods for system design and wireless networks.