T2FA: A Heuristic Algorithm for Deadline-constrained Workflow Scheduling in Cloud with Multicore Resource

Zaixing Sun, Chonglin Gu*, and Hejiao Huang
Harbin Institute of Technology (Shenzhen), China
Email: guchonglin@hit.edu.cn

Honglin Zhang
Shandong University, China

2021 IEEE 14th International Conference on Cloud Computing (CLOUD) | 978-1-6654-0060-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/CLOUD53861.2021.00048

***Abstract*—Workflow scheduling is one of the most challenging problems in cloud computing. This paper proposes a heuristic algorithm task type first algorithm (T2FA) for solving deadline-constrained workflow scheduling in cloud with multicore resource (DWS_CMR). The objectives to be minimized are the maximal completion time (i.e., makespan) and the total costs. Firstly, resource model and workflow application model are introduced. Resource model has the configurations of multicore, processing capacity, bandwidth and leasing price, and workflow application model is described by directed acyclic graph (DAG). Based on above models, the mathematical model of DWS_CMR is established, which allows multiple tasks to run concurrently on multicore resources. Secondly, to exploit the characteristics of the problem, the structures of DAG are decomposed and formulated. Merging tasks conforming to the first structure into task blocks can simplify DAG. Four special types of tasks are extracted from the second and third structures, and are preferentially scheduled in task scheduling stage. Then, a new interrelated calculation method of estimated start time and actual start time of tasks is proposed, which can complete the task-to-resource mapping. Finally, T2FA is devised, which incorporates two important phases, including pre-processing and task scheduling. Experimental results show that T2FA can achieve significantly better schedules in most test cases compared to several existing algorithms.**

***Keywords*—Cloud Computing, Multicore Resource, Workflow Scheduling, Deadline Constraint, Directed Acyclic Graph**

I. Introduction

CLOUD computing is emerging as a primary computing paradiam that is researched, developed, and deployed by enterprises, governments, and academic institutes in recent years [1]. By supporting a pay-as-you-go service model, cloud computing removes initial capital, maintenance, and software licensing cost [2]. In addition, it has greatly changed the way information technology infrastructure is provided to satisfy various business needs by enabling on-demand infrastructure provisioning.

For the execution scenarios of tasks in cloud, there are two categories, Non-multiprogrammed, where a single computing resource processes a single task, and Multiprogrammed, where a single computing resource processes multiple tasks at the same time. Among them, the Non-multiprogrammed method has been fully studied [3]. If a task cannot fully use the computing resources provided by a service, having it exclusively occupying the entire service will lead to resource over-provisioning. With the development of multicore technology, the computing resources provided by cloud service providers are mainly in the form of virtual machine (VM). In the research of [4]–[6], each task is allowed to occupy one CPU core instead of the whole service. In real workflow, the demands of tasks may not only include processors, but also many other types of resources, such as memory and I/O bandwidth. Therefore, the research of multiprogrammed task execution model becomes more and more important.

Workflow scheduling problem is a typical NP-Complete problem, which has been a hot research topic in the field of distributed computing for many years [7]–[9]. Workflow is usually described by directed acyclic graph (DAG), in which each node represents a task, and the directed relationship between two node pairs represents the processing constraint of the tasks. The essence of workflow scheduling is to establish a set of effective mapping relationships from task to VM in cloud, and to minimize the maximum completion time (Makespan) and cost under the constraints of Quality of Service (QoS), so as to achieve efficient utilization and balanced allocation of system resources. In recent years, deadline constraint workflow scheduling is becoming increasingly important in cloud computing. Missing the deadline may result in severe losses especially for large-scale scientific computing and the Internet services with widespread influence.

Workflows have been proved to be an effective and popular method to model various scientific computing problems in parallel and distributed systems, such as larges-cale scientific problems in the fields of astronomy, bioinformatics, and physics [10]. Scientific workflows may vary in size from a few tasks with limited resource needs to millions of tasks requiring tens of thousands of processing hours, terabytes of storage and high bandwidth network resources. Such complex workflows have ever-increasing data and computing demands, so they need a high-performance computing environment to be executed in a reasonable time [11]. The appearance of the infrastructure as a service (IaaS) clouds offers us a new utility-based platform to execute large scale workflows.

In cloud computing environment, the distributed scheduling system, which is the basic technology to maintain the efficient operation of its data center, mainly includes two key modules: resource management and task scheduling. The quality of

2159-6190/21/$31.00 ©2021 IEEE
DOI 10.1109/CLOUD53861.2021.00048

scheduling technology directly affects the user satisfaction and the efficiency of cloud computing system. Optimizing scheduling technology can effectively reduce the completion time of tasks, reduce energy consumption and save operating costs, and can effectively improve the utilization of computing resources. Although there are many existing workflow scheduling algorithms in the traditional distributed or heterogeneous computing environment, because the cloud is different from the traditional heterogeneous environment, its service-based resource management method and on-demand pricing strategy make the existing workflow scheduling algorithms have limitations in the cloud environment. Therefore, it is necessary to develop and study workflow scheduling technology in cloud environment, which has theoretical significance and practical value.

This paper studies deadline-constrained workflow scheduling in cloud with multicore resource (DWS_CMR) of minimizing the maximal completion time (i.e., makespan) and the total costs (TC). In DWS_CMR, tasks are executed on a certain number of heterogeneous multicore VMs. The VMs are acquired and released dynamically. The bandwidth of VM needs to be considered when transferring data between tasks. The challenges of this work mainly consider three aspects:

- The strong inter-dependencies between workflow tasks. When adjacent tasks are executed on different VMs, the data transmission time is different due to different bandwidths, which affects the starting execution time of subsequent tasks.
- The task should not only select the appropriate VM, but also specify the core of the selected VM. In addition, the execution time of tasks on VMs with different configurations is different. The higher the configuration of VM, the shorter the execution time of task, and the higher the price per unit time of VM lease.
- The deadline constraint is satisfied as much as possible.

Based on the above challenges, this paper proposes a heuristic algorithm called task type first algorithm (T2FA) to solve DWS_CMR.

The main contributions of this paper are as follows.

- A new workflow scheduling model is presented, which allows multiple tasks to run concurrently on multicore resources.
- The structures of DAG are decomposed and formulated. Different structures act on different stages of the algorithm.
- An integrated method is designed to calculate the estimated start time and actual start time of a task. When the actual start time is determined, the mapping from task to resource is completed.
- A heuristic algorithm T2FA is proposed to solve DWS_CMR. Extensive simulation experiments demonstrate the superiorities and competitiveness of T2FA.

The rest of this paper is organized as follows. Section II reviews the related work. Section III presents the cloud workflow scheduling model, including resource model, application model, and workflow scheduling. Section IV provides details of the designs for T2FA to address DWS_CMR. Section V presents the experimentation and evaluation. Finally, Section VI concludes the paper.

II. Related Work

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

In homogeneous resource environment, all computing resources have the same configuration of CPU, memory, etc. Byun et al. [12] proposed a Partitioned Balanced Time Scheduling (PBTS) algorithm, which estimates the minimum number of computing hosts required to execute a workflow under user deadline, minimizing the gross cost during the entire application lifetime. Wu et al. [13] proposed a two-stage method VM instances and VM instance hour minimization for deadline constrained DAG applications deployed on computer clouds. Firstly, the minimal slack time and minimal distance algorithm was designed to find the minimum number of VM instances needed to guarantee the deadline and minimize the makespan. On this basis, VM instance hour minimization algorithm was used to reduce the number of instance hours, thus reducing the cost.

In heterogeneous scenes, Abrishami et al. [14] proposed two algorithms called IaaS Cloud Partial Critical Paths (IC-PCP) and IC-PCP with Deadline Distribution (IC-PCPD2) for minimizing the cost of workflow execution under deadline constraints. The former distributes the overall deadline to PCPs, while the latter further distributes the deadline to each task in proportion to its minimum execution time. Then, the cheapest resource which can meet the latest finishing time of the task is selected for partitions and tasks, respectively. The simulation results showed that both algorithms have a promising performance, with IC-PCP performing better than IC-PCPD2 in most cases. Rodriguez et al. [11] 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. [15] 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.

With the development of multicore processor technology, Deldari et al. [16] established a heterogeneous computing resource model, in which each multicore processor consists of several homogeneous cores, and proposed a cluster combining algorithm (CCA), to minimize the execution cost while

meeting the deadline constraints submitted by users. Zhu et al. [17] 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 the best 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 (DyDL) was proposed to minimize the cost of workflow execution.

III. Cloud Workflow Scheduling Model

In this section, the resource model and workflow application model are described, and then the workflow scheduling mathematical model is established.

A. Resource Model

Elastic compute service platforms deliver computing resources to customers via virtual machine instances. The VMs are heterogeneous with various configurations (e.g. CPU numbers, processing capacity and network bandwidth) and can be defined as $VM = (P, N, U, B, M)$.

- $P = \{P_i | i = 1, 2, \cdots, m\}$ is the set of VMs.
- $N = \{N_i | i \in P\}$ is the set of CPU cores, where N_i is the number of CPU cores in VM P_i. When a task is executed on a multicore CPU, it can occupies one CPU core at most.
- $U = \{U_i | i \in P\}$ is the set of processing capacity of VM, where U_i is the processing capacity of VM P_i. The higher the processing capacity of VM, the shorter the execution time of a task.
- $B = \{B_{ij} | i, j \in P\}$ is the set of the data transfer rate between VMs. $B_{ij} = \min\{b_i, b_j\}$ is the data transfer rate between VM P_i and P_j, where $B_{ij} \to +\infty$ when $i = j$ and b_i is bandwidth for VM P_i.
- $M = \{M_i | i \in P\}$ is the set of the leasing price, where M_i is the per-unit price of VM P_i. The pricing model is based on a pay-as-you-go billing scheme and the users are charged for the number of billing intervals of the VMs they lease, even if these VMs have not been completely used in the last billing interval. The billing interval is specified by the cloud provider. For example, Amazon EC2 platform sets 1 hour as a minimum unit of lease time while Microsoft Azure sets 1 minute. Let CU denote the billing interval of the cloud service platform.

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 assumes that the VMs have enough memory to perform workflow tasks.

B. Workflow Application Model

The user submits a workflow to the cloud service platform. Each task of the workflow has a fixed resource requirement. Workflow is represented by direct acyclic graph (DAG) $G = (T, W, E, D, C)$, which can describe different scientific and engineering problems. These five elements in DAG are introduced separately below.

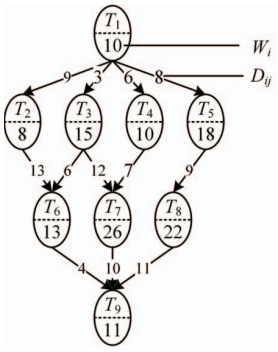


Fig. 1. A sample workflow

- $T = \{T_i | i = 1, 2, \cdots, n\}$ is the set of all tasks, where T_i represents a task in the DAG and n represents the total number of tasks.
- $W = \{W_i | i \in T\}$ is the set of weights on the task and represents the reference execution time of tasks , in seconds. $ET_{ik} = \frac{W_i}{U_k}$ is the execution time of a task T_i on VM P_k.
- $E = \{E_{ij} = (T_i, T_j) | T_i, T_j \in T; i < j\}$ is the set of dependencies between tasks. Dependency (T_i, T_j) indicates a precedence constraint between tasks T_i and T_j, where T_i is the direct predecessor task of T_j and T_j is the direct successor task of T_i.
- D_{ij} is the amount of data transferred from task T_i to task T_j, in bytes.
- $C = \left\{C_{ij}^{kl} | i, j \in T; k, l \in P; i < j\right\}$ is the set of data transfer time between tasks with dependency. $C_{ij}^{kl} = \frac{D_{ij}}{B_{kl}}$ represents the communication time between task T_i and task T_j, where task T_i is executed on VM P_k and task T_j is executed on VM P_l. When $k = l$, the transmission time on the same VM is 0.

A sample workflow is shown in Fig. 1. Each node represents a task and the value under the node is the weight of the task. Each arc represents the processing constraints between tasks, and the value on it represents data transmitted.

In addition, each workflow has a deadline DL, which is defined as the time limit specified by the user for the execution of the workflow.

C. Workflow Scheduling

Workflow scheduling is to schedule the tasks of the workflow to the VMs on the cloud platform. In essence, workflow scheduling establishes a mapping between tasks and VMs. Fig. 2 shows a sample schedule generated for the workflow shown in Fig. 1. Each task is mapped onto one of the two available VMs. For convenience of observation, the two VMs selected have the same configuration, where $N_1 = N_2 = 2$, $U_1 = U_2 = 1$, $B_{12} = B_{21} = 1$.

This work focuses on finding a schedule to execute a workflow on an IaaS cloud such that total execution cost

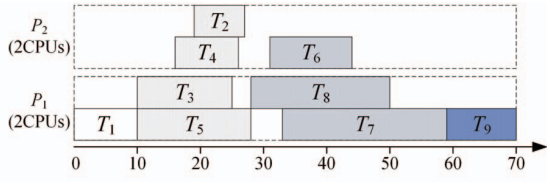


Fig. 2. Workflow scheduling

and makespan are minimized while meeting the user defined deadline constraint. A schedule is denoted as $\Pi = (\gamma, \delta, \alpha, \beta)$ in terms of a task to VM and core mapping, and the actual start and finish time of tasks.

- $\gamma = \{\gamma_i | i \in T\}$ is the mapping of tasks to VMs, where γ_i indicates that task T_i is assigned to be executed on VM P_{γ_i}.
- $\delta = \{\delta_i | i \in T\}$ is the set of cores corresponding to tasks, where δ_i indicates that task T_i is assigned to be executed on the $\delta_i th$ core of VM P_{γ_i}.
- $\alpha = \{\alpha_i | i \in T\}$ and $\beta = \{\beta_i | i \in T\}$ are the set of actual start time and completion time of task respectively.

In addition, the lease start time of a VM is the earliest start time on the VM, and the lease end time is the latest finish time on the VM. The lease time of VM P_k is shown in Eq. 1.

$$\tau_k = \max\{\beta_j\} - \min\{\alpha_i\}, \ i, j \in T, \ \gamma_i = \gamma_j = k. \quad (1)$$

The objective of scheduling is $f(\Pi) = (C_{max}, TC)$. Based on the previous definitions, DWS_CMR can be formally defined as follows:

$$\textbf{Minimize} \quad C_{max} = \max\{\beta_i | i \in T\}, \quad (2)$$

$$\textbf{Minimize} \quad TC = \sum_{k=1}^{m} M_k \times \left\lceil \frac{\tau_k}{CU} \right\rceil, \quad (3)$$

$$\textbf{subject to} \quad C_{max} \leq DL, \quad (4)$$

$$\beta_i \leq \alpha_j, \quad (T_i, T_j) \in E. \quad (5)$$

IV. The Proposed Workflow Scheduling Algorithm

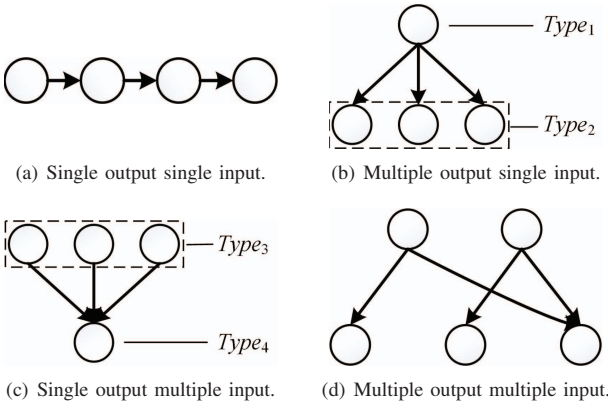
This section will detail the proposed task type first algorithm (T2FA) for workflow scheduling after explaining DAG structure decomposition, task topological level, task available start time and actual start time.

A. 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.

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

$$|Suc(T_i)| = \sum_{T_j \in Suc(T_i)} |Pre(T_j)| = 1, \quad (6)$$



(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

where $Suc(T_i) = \{T_j | (T_i, T_j) \in E\}$ represents the set of all direct successors of task T_i and $Pre(T_j) = \{T_i | (T_i, T_j) \in E\}$ represents the set of all direct predecessors of task T_j.

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 the solution complexity and solution space can be reduced to the same extent.

- 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.

$$|Suc(T_i)| = \sum_{T_j \in Suc(T_i)} |Pre(T_j)| > 1. \quad (7)$$

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

$$Type_1 = \{T_i | T_i \text{ satisfies Eq. } 7\}, \quad (8)$$

$$Type_2 = \{T_j | T_j \text{ satisfies Eq. } 7\}. \quad (9)$$

- 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.

$$|Pre(T_j)| = \sum_{T_i \in Pre(T_j)} |Suc(T_i)| > 1. \quad (10)$$

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

$$Type_3 = \{T_i | T_i \text{ satisfies Eq. } 10\}, \quad (11)$$

$$Type_4 = \{T_j | T_j \text{ satisfies Eq. } 10\}. \tag{12}$$

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

B. Task Topological Level

Given a DAG-based workflow, its task ti's topological level $Lev(T_i)$ is defined as [13]:

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

Thus, the set $LT_{jk} \in LT$ of task in each level can be obtained, as shown in Eq. 14, where LT_{jk} represents the kth task of the jth level, $j = 0, 1, \cdots, \max\{Lev\}$, $k = 1, \cdots, |LT_j|$, and $|LT_j|$ is the number of tasks in jth level.

$$LT_j = \{T_i | j = Lev(T_i), i = 1, 2, \cdots, n\}. \tag{14}$$

In particular, tasks at a lower topological level have higher priorities than tasks at a higher level [13].

C. Available Start Time and Actual Start Time

Available start time (AT) refers to the earliest start time when task is assumed to be executed on a specified VM core, which is determined by the actual completion time of the predecessor task and the completion time of the VM core. For example, AT_{ikr} represents the available start time of task T_i on the rth core of VM P_k. It is obtained as given in Eq. 15:

$$AT_{ikr} = \begin{cases} \max\{0, PC_{kr}\}, & if\ Pre(T_i) = \phi \\ \max\left\{ \begin{matrix} \max\limits_{T_j \in Pre(T_i)} \{\beta_j + C_{ij}^{kl}\}, \\ PC_{kr} \end{matrix} \right\}, & otherwise. \end{cases} \tag{15}$$

Where $l = \gamma_j$ is the assigned VM of task T_j, and PC_{kr} is the completion time on the rth core of VM P_k.

The actual start time of task T_i (i.e. α_i) is determined by AT, MFT and $setP$. MFT is the current maximum finish time, and $setP$ is the set of VMs that have assigned tasks. In the set $setP$, if the smallest AT is less than MFT, α_i, the corresponding AT is taken as the α_i; otherwise, select the smallest in P as the α_i. While obtaining the actual start time α_i, the VM γ_i, and core δ_i that execute the task can be obtained. The actual completion time of task T_i (i.e. β_i) is equal to the sum of actual start time and execution time. See Algorithm 2 for details.

For VM selection, VM is first selected from the VMs with assigned tasks, which can make the scheduling more compact; when the current completion time constraint is not met, selecting from all available VMs can make the task start processing as early as possible.

Algorithm 1: T2FA

Input: Resource (P, N, U, B, M), workflow (T, E, W, C, D), $Deadline$
Output: $\Pi = (\gamma, \delta, \alpha, \beta)$, $f(\Pi) = (C_{max}, TC)$

1 Simplify DAG by Eq. 6;
2 Compute LT using Eqs. 13 and 14;
3 $v' = \max\{|LT_0|, |Suc(LT_{0i})| \, | i = 1, 2, \cdots, |LT_0|\}$;
4 **if** $\{N_i | N_i \geq v', i \in P\} \neq \phi$ **then**
5 $\quad v = \arg\min\{N_i | N_i \geq v', i \in P\}$;
6 **else**
7 $\quad v = \arg\max\{N_i | i \in P\}$;
8 **end**
9 $setP = \{v\}$, $MFT = \max\{W_i | T_i \in LT_0\} / U_v$;
10 **for** $l \leftarrow 0$ **to** $\max\{Lev\}$ **do**
11 $\quad$ Random generation of a ranking from 1 to 4, denoted by RCS ;
12 $\quad$ **foreach** $c \in RCS$ **do**
13 $\quad\quad dt = LT_l \cap Type_c$;
14 $\quad\quad \pi = Type_c - dt$;
15 $\quad\quad Type_c = Type_c - dt$;
16 $\quad\quad LT_l = LT_l - dt$;
17 $\quad\quad$ Sort the tasks in π in descending order of weight value;
18 $\quad\quad$ call $TaskSchedule(\pi)$;
19 $\quad$ **end**
20 $\quad$ Sort the unscheduled tasks in LT_l in descending order of weight value;
21 $\quad$ call $TaskSchedule(LT_l)$;
22 **end**
23 Compute objectives.

D. Task Type First Algorithm (T2FA)

The detail of the proposed T2FA is listed as Algorithm 1. T2FA is designed according to the characteristics of resource model and workflow application model. In the multicore resource model, the virtual machines are heterogeneous, while the different cores in a virtual machine are homogeneous. In the DAG description of the workflow application model, there are processing constraints between different layers, and tasks in the same layer can be processed in parallel.

Pre-processing. Through the structural decomposition of DAG in Section IV-A, DAG can be simplified by Eq. 6 (line 1 in Algorithm 1). Then divide the task topological level and count the tasks of each level (line 2 in Algorithm 1). Since it is a multicore VM, the choice of the first VM is crucial. The algorithm considers three factors: the number of tasks in 0th level (i.e. $|LT_0|$), the maximum number of subsequent tasks and the number of VM cores, which can reduce data transmission to a certain extent (lines 3-8 in Algorithm 1). On this basis, set the set of candidate VMs $setP$ and the expected maximum finish time value MFT (line 9 in Algorithm 1).

Task scheduling. In the task scheduling stage, it is divided into three levels according to the topology level and task type. Level 1 is the topological level from low to high (line 10 in

Algorithm 2: TaskSchedule(π)

```
1  foreach i ∈ π do
2      Compute AT_ikr using Eq. 15, ∀ k ∈ P, r ∈ N;
3      k, r = arg min {AT_ikr | k ∈ setP, r ∈ N} ;
4      if AT_ikr + ET_ik > MFT then
5          k, r = arg min {AT_ikr | k ∈ P, r ∈ N} ;
6      end
7      γ_i = k, δ_i = r, α_i = AT_ikr;
8      β_i = α_i + ET_ik;
9      if β_i > MFT then MFT = β_i;
10     if k ∉ setP then setP = setP ∪ {k};
11 end
12 return
```

Algorithm 1), level 2 is the four special task types (lines 11-19 in Algorithm 1), and level 3 is the general type of tasks (lines 20-21 in Algorithm 1). Four special task types are scheduled in random order. Within the same type, tasks are arranged in descending order of task weight, and then scheduled according to Algorithm 2. Algorithm 2 is the process of getting the actual start time of tasks and allocating VMs, as introduced above in Section IV-C. The expected maximum finish time MFT in the algorithm is used as a reference value, and its initial value is set in lines 3-9 of Algorithm 1. When scheduling tasks, the VM whose existing tasks are executed is preferentially selected. When the available finish time of the task is less than MFT, deploy the task to the VM with the earliest available finish time; otherwise, select the VM with the earliest available finish time from all VMs and update MFT (lines 3-6 and line 9 in Algorithm 2). The purpose of setting MFT is to make scheduling more compact.

The **main ideas** of T2FA are as follows. Firstly, through DAG structure decomposition, DAG is simplified and tasks are divided into different types of sets (lines 1-2 in Algorithm 1). Then, the tasks are scheduled in turn according to the topology level (line 10 in Algorithm 1). At each level, tasks in Type1-4 are scheduled preferentially (lines 11-12 in Algorithm 1). Among the tasks of the same type, the task with the longest execution time is preferentially scheduled (lines 13-18 in Algorithm 1).

V. Experimental Results and Discussion

A. Experimental Environment

In the experiment, we use 5 representative VM types from low configuration to high configuration. Among them, the CPU core number and unit price are configured by Amazon EC2's C4-type VMs, as shown in Table 1. Correspondingly, the following two parameters are set according to different VM configurations. According to the research in [10], [18], the processing capacity of VM is expressed by CPU clock frequency, in GHz. The processing capacity $U = \{U_1, U_2, U_3, U_4, U_5\} = \{1.8, 2.2, 2.7, 3.15, 3.5\}$ and the bandwidth $b = \{b_1, b_2, b_3, b_4, b_5\} = \{1, 1.5, 2, 3, 3\}$. The unit of bandwidth is Gbps. The number of virtual machines used in the experiment is $m = \{5, 10\}$. When $m = 5$, each type of VM uses one; when $m = 10$, each type of VM uses two. The billing interval is 1 hour, that is, $CU = 3600$s.

TABLE I
Configurations and Prices of Virtual Machines[1]

	VM Type	vCPU	Cost ($/h)
1	c4.large	2	0.1
2	c4.xlarge	4	0.199
3	c4.2xlarge	8	0.398
4	c4.4xlarge	16	0.796
5	c4.8xlarge	36	1.591

Five real-world workflow types from different scientific areas are used in the experiments: CyberShake, Epigenomics, Inspiral, Montage and Sipht [14], [19], [20]. Each of them has different structures and characteristics and is widely used to evaluate the performance of the workflow scheduling approaches. Fig. 4 shows the sample DAG structures of these workflow [21]. Their specific characteristics are as follows. More details about these workflows can be found in [19].

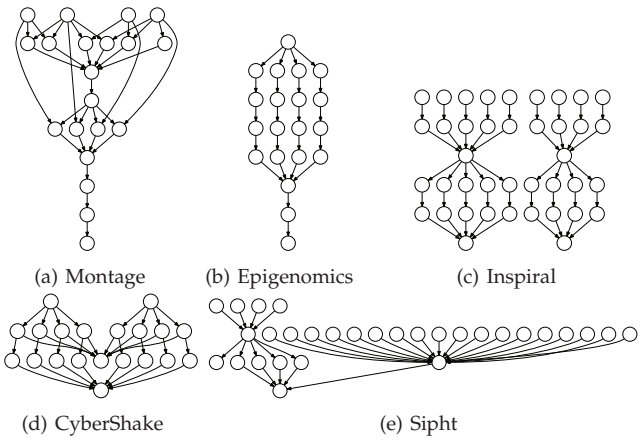


Fig. 4. Structures of the five real-world workflow

- *CyberShake*: It is an earthquake science application used to characterize earthquake hazards through combining large datasets.After decomposition, it mainly includes three structures: SOSI, MOSI and MOMI in Fig. 3.
- *Epigenomics*: It is a highly pipelined biology application which maps the epigenetic state of human cells.After decomposition, it mainly includes three structures: SOSI, MOSI and SOMI in Fig. 3.
- *Inspiral*: It is used in the physics field to detect gravitational waves. After decomposition, it mainly includes three structures: SOSI, MOSI and SOMI in Fig. 3.
- *Montage*: It is used in astronomy with the aim of constructing the desired mosaic of the sky on the basis of input images. After decomposition, it mainly includes two structures: SOSI and SOMI in Fig. 3.

[1] https://aws.amazon.com/ec2/pricing/on-demand/

- *Sipht*: It is used for automating the search of sRNA encoding-genes for bacterial replicons from bioinformatics. After decomposition, it mainly includes two structures: SOMI and MOMI in Fig. 3.

The five real-world workflows can be decomposed into at least two structures in Fig. 3, and all of them have SOMI structure (Fig. 3(c)). All these workflow instances are generated in form of Directed Acyclic Graph in XML (DAX) format by Pegasus WorkflowGenerator [15], [19], [22], [23], and are publicly available on Pegasus website[2]. 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. The number of tasks varies from 24-1000. Using 'Workflow type_Number of tasks' to distinguish different problems, such as CyberShake_30 represents CyberShake of 30 tasks.

B. Deadline Setup

Deadline of workflow is an important constraint of DWS_CMR, if the deadline is generously relaxed, there is enough slack time to accommodate for the VM acquisition delay and the performance variation. Therefore, a comprehensive evaluation requires performance analysis on all possible deadlines. To this end, use the following rules to set deadline.

First, estimate maximum execution time (MET) and the maximum transmission time (MTT):

$$MET_i = \max\left\{ET_{ik} | P_k \in P\right\}, \tag{16}$$

$$MTT_{ij} = \max\left\{\frac{D_{ij}}{b_k}\middle| P_k \in P\right\}. \tag{17}$$

Then estimate start time (EST) and finish time (EFT):

$$EST_i = \begin{cases} 0, & if\ Pre(T_i) = \phi \\ \max\limits_{T_j \in Pre(T_i)} \left\{EFT_j + MTT_{ji}\right\}, & otherwise. \end{cases} \tag{18}$$

$$EFT_j = EST_j + MET_j. \tag{19}$$

The deadline is set to:

$$DL = \mu \times \max\left\{EFT_i | T_i \in T\right\}, \tag{20}$$

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

C. Baseline Algorithms

To illustrate the effectiveness of the proposed algorithm, three baseline algorithms are implemented for comparison, including two heuristic algorithms IC-PCP [14] and JIT-C [15], and a meta-heuristic algorithm PSO [11]. These three algorithms have been introduced in Section II and are classical algorithms in solving the problem of a cost-minimization, deadline-constrained workflow scheduling problem. Although originally proposed for non-multicore resource problem, they can be applied to DWS_CMR with some modifications. The parameters of PSO are set according to the optimal parameters given in [11], which are $c_1 = c_2 = 2.0$, $\omega = 0.5$, and the number of particles and iteration times are set to 100. All algorithms are coded in Python and are executed on Intel Core i5-9500 3.0GHz processor with 32GB RAM.

[2]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowHub

D. 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: makespan $Cmax$, total cost TC and running time of the algorithms. $Cmax$ and TC are two objectives of DWS_CMR, and are normalized by the following formula:

$$f_{norm} = \frac{f - f_{min}}{f_{max} - f_{min}}, \tag{21}$$

where f_{min} and f_{max} are the minimum and maximum value achieved among all the four algorithms compared, respectively. Running time is the CPU execution time for the algorithm to obtain the scheduling solution for a given problem. Although DWS_CMR is static scheduling problem, in order to provide a practical solution, running time of is the key constraint. This metric basically gives the average cost of each algorithm [8].

When the solution obtained by the algorithm cannot satisfy the deadline constraint (i.e. infeasible solution), the values of the three metrics are null.

1) *Comparison of* 5 *VMs.* Fig. 5 and Fig. 6 show the comparison of makespan and cost under different deadline factors when the number of VMs is 5. For makespan, in terms of CyberShake, Epigenomics, and Inspiral workflows, PSO can get the smallest makespan in terms of minimum scale, followed by T2FA. With the expansion of scale, T2FA has become more prominent and gradually replaced PSO. Among Montage and Sipht workflows, T2FA can always get the smallest makespan except for Montage workflow with 1000 tasks.

For cost, T2FA can always get a relatively small cost, except for Sipht workflow. For Sipht workflow, T2FA has medium performance on small and medium-sized problems and better performance on 1000 tasks workflow.

For the workflow with 1000 tasks, when deadline factor $\mu = 0.8$, only JIT-C and T2FA can get feasible solutions on Sipht workflow, and T2FA is better than JIT-C. With the increase of deadline factor, other algorithms can get feasible solutions for each type of workflow problem, which further shows the effectiveness of deadline setting method.

In addition, IC-PCP algorithm can't get the solution that satisfies the deadline constraint in all workflow types when deadline factor $\mu = 0.8$, which is related to the setting method of deadline and the characteristics of the algorithm, not the weak performance of the algorithm. Moreover, IC-PCP can also obtain smaller cost solutions for medium scale problems. It also shows that IC-PCP algorithm obtains the least cost at the expense of time.

For PSO, in small and medium-sized CyberShake, Epigenomics, Inspiral and Sipht workflow problems, it can always get relatively small makespan, but it always gets the highest

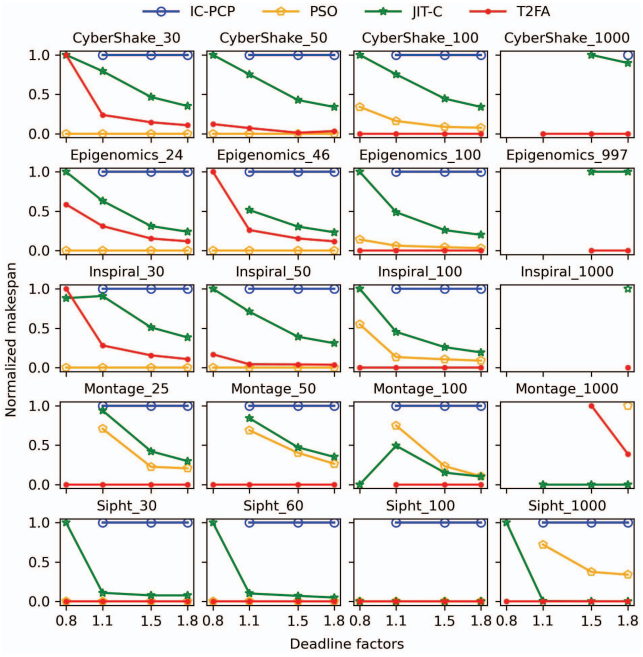


Fig. 5. The normalized makespan of scheduling workflows with IC-PCP, PSO, JIT-C and T2FA with $m = 5$.

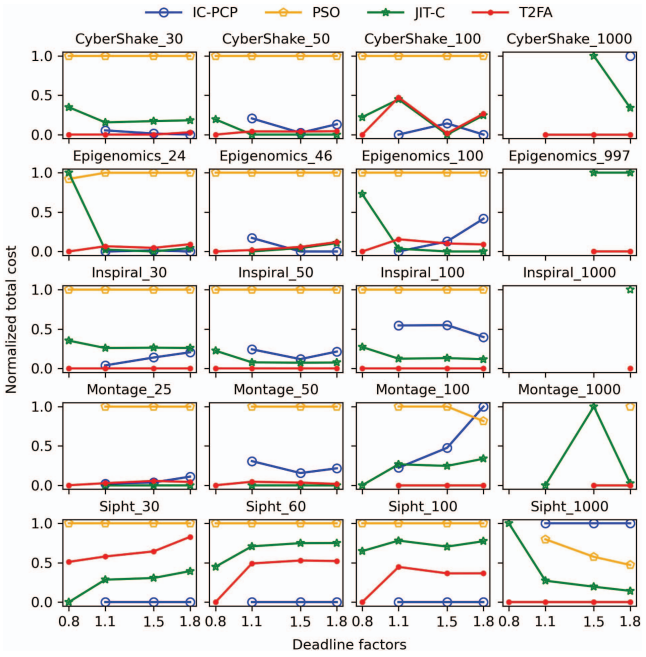


Fig. 6. The normalized total cost of scheduling workflows with IC-PCP, PSO, JIT-C and T2FA with $m = 5$.

cost in almost all problems, which indicates that PSO gets the least time at the expense of cost.

For JIT-C algorithm, makespan and cost are always at the second or third level.

In general, T2FA can take into account both makespan and cost, and can obtain a better solution.

2) *Comparison of* 10 *VMs.* Fig. 7 and Fig. 8 show the comparison of makespan and cost under different deadline factors when the number of VMs is 10. Except for the large-scale problem of 1000 task, the results of other problems are similar to the previous experiments. Because there are fewer algorithms to obtain infeasible solutions for large-scale problems.

It is worth mentioning that the number of workflows in which the JIT-C algorithm obtains infeasible solutions has increased, such as CyberShake for 1000 tasks, CyberShake for 30 tasks at $\mu = 0.8$, Epigenomics for 24 tasks, and Inspiral for 30 tasks. The main reason is that JIT-C algorithm adopts the cheapest rule in the VM allocation stage, which is also a time-consuming rule to some extent. Especially in the selection of the first VM, it is often the VM with the lowest configuration, which will lead to a lot of data transmission costs.

3) *Running Time of the Algorithms.* Table II 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 T2FA is always the least and the longest is no more than 1 second. With the increase of the scale of the problem, the running time does not change significantly. Since PSO is a swarm intelligence algorithm, through iterative optimization, it is conceivable that the running time of the algorithm is longer than that of the heuristic algorithm. 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 II
AVERAGE RUNNING TIME OF SCHEDULING ALGORITHMS (IN SECOND).

	$m = 5$				$m = 10$			
	IC-PCP	PSO	JIT-C	T2FA	IC-PCP	PSO	JIT-C	T2FA
CyberShake_30	2.849	99.388	0.372	**0.014**	3.282	127.606	0.409	**0.025**
CyberShake_50	3.577	171.059	1.014	**0.025**	4.372	219.290	1.037	**0.039**
CyberShake_100	6.434	386.915	4.293	**0.052**	7.383	419.825	4.685	**0.060**
CyberShake_1000	42.124	*null*	396.285	**0.596**	50.950	*null*	*null*	**0.700**
Epigenomics_24	2.265	64.712	0.038	**0.009**	2.704	85.990	0.037	**0.008**
Epigenomics_46	2.739	126.570	0.115	**0.018**	3.248	168.306	0.113	**0.018**
Epigenomics_100	4.499	278.127	0.341	**0.034**	6.023	344.732	0.342	**0.051**
Epigenomics_997	*null*	*null*	26.123	**0.366**	21.585	*null*	26.056	**0.400**
Inspiral_30	2.761	81.205	0.113	**0.011**	3.242	114.870	0.113	**0.014**
Inspiral_50	3.470	137.190	0.272	**0.021**	3.814	185.900	0.261	**0.025**
Inspiral_100	4.148	276.147	0.969	**0.036**	4.497	348.541	0.958	**0.043**
Inspiral_1000	*null*	*null*	88.512	**0.426**	26.374	*null*	85.445	**0.434**
Montage_25	3.014	87.299	0.208	**0.012**	3.435	99.022	0.211	**0.016**
Montage_50	4.967	191.874	0.951	**0.030**	5.236	214.839	1.151	**0.037**
Montage_100	7.913	406.798	4.139	**0.064**	8.356	507.725	4.464	**0.147**
Montage_1000	*null*	4695.863	488.354	**0.714**	294.147	4686.436	476.947	**0.825**
Sipht_30	2.755	89.851	0.287	**0.012**	2.885	109.626	0.282	**0.014**
Sipht_60	2.796	173.485	1.040	**0.023**	3.172	218.842	1.083	**0.035**
Sipht_100	4.058	270.520	2.776	**0.038**	4.348	335.639	3.055	**0.060**
Sipht_1000	20.819	2931.188	275.781	**0.512**	22.586	3412.730	282.565	**0.638**
Avg	7.129	615.776	64.599	**0.151**	24.082	682.348	46.801	**0.179**

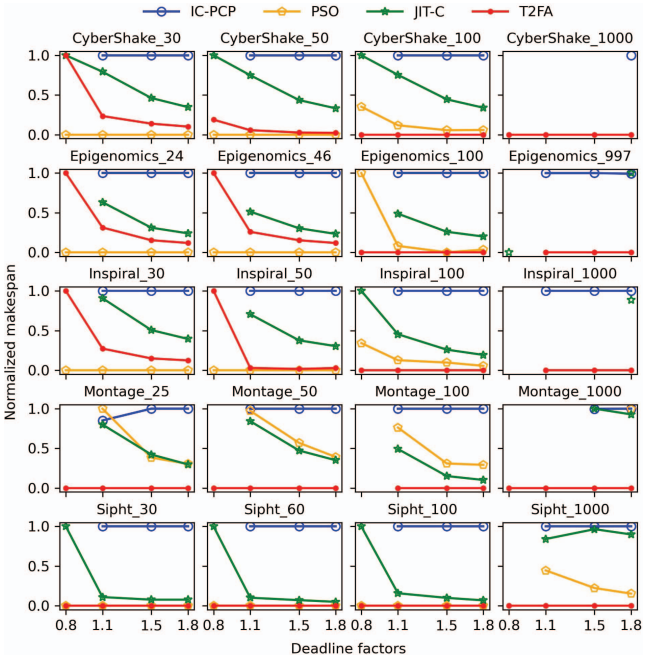


Fig. 7. The normalized makespan of scheduling workflows with IC-PCP, PSO, JIT-C and T2FA with $m = 10$.

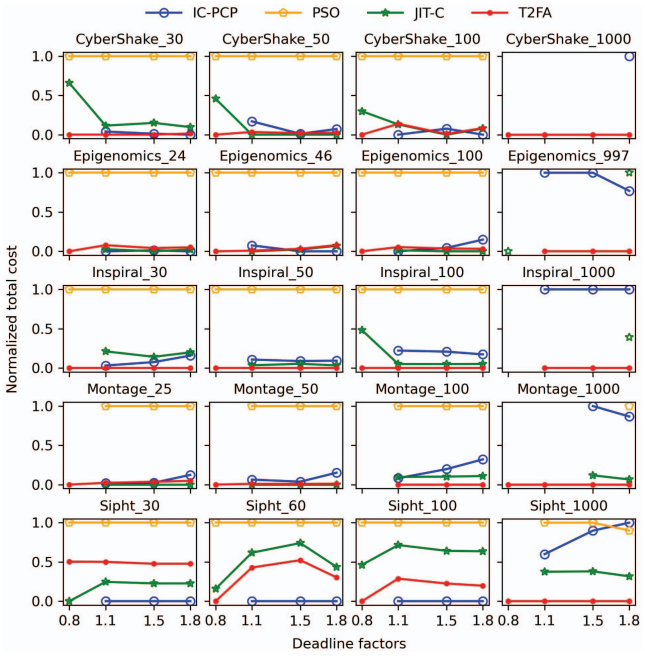


Fig. 8. The normalized total cost of scheduling workflows with IC-PCP, PSO, JIT-C and T2FA with $m = 10$.

VI. Conclusion

This paper presented a heuristic algorithm task type first algorithm (T2FA) for solving deadline-constrained workflow scheduling in cloud with multicore resource (DWS_CMR), which minimizes the maximal completion time (i.e., makespan) and the total costs (TC). Firstly, the relevant models of DWS_CMR were introduced. Secondly, the structure of directed acyclic graph (DAG) description of workflow was decomposed, and three structures were emphatically analyzed. The first structure was used to simplify DAG, and the other two structures contained four special types of tasks, which were used for task scheduling. Then, the definitions of available start time and actual start time were introduced, and the resource allocation method was determined. A novel and effective T2FA was further proposed to solve DWS_CMR. Finally, the simulation results and comparisons demonstrated that T2FA outperforms the baseline algorithms includings two heuristic algorithms IC-PCP and JIT-C, and a meta-heuristic algorithm PSO. In the future, our work is to investigate the applications of the presented method to the multiobjective workflow scheduling problems with energy consumption.

VII. Acknowledgements

This work is financially supported by National Key R&D Program of China under Grant No.2017YFB0803002, National Natural Science Foundation of China under Grant No.61732022, and Guangdong Basic and Applied Basic Research Foundation under Grant No.2019A1515110214.

References

[1] H. Yuan, J. Bi, and M. C. Zhou, "Energy-Efficient and QoS-Optimized Adaptive Task Scheduling and Management in Clouds," *IEEE Transactions on Automation Science and Engineering*, pp. 1–12, 2020.

[2] S. G. Domanal, R. M. R. Guddeti, and R. Buyya, "A Hybrid Bio-Inspired Algorithm for Scheduling and Resource Management in Cloud Environment," *IEEE Transactions on Services Computing*, vol. 13, no. 1, pp. 3–15, 2020.

[3] M. Wieczorek, A. Hoheisel, and R. Prodan, "Towards a general model of the multi-criteria workflow scheduling on the grid," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 237–256, 2009.

[4] S. Smanchat and K. Viriyapant, "Taxonomies of workflow scheduling problem and techniques in the cloud," *Future Generation Computer Systems*, vol. 52, pp. 1–12, 2015.

[5] Y. C. Lee, H. Han, A. Y. Zomaya, and M. Yousif, "Resource-efficient workflow scheduling in clouds," *Knowledge-Based Systems*, vol. 80, pp. 153–162, 2015.

[6] R. Pathan, P. Voudouris, and P. Stenström, "Scheduling Parallel Real-Time Recurrent Tasks on Multicore Platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 4, pp. 915–928, 2018.

[7] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427–1436, 1989.

[8] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[9] 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," *Journal of Systems and Software*, vol. 133, pp. 1–16, 2017.

[10] 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 Transactions on Services Computing*, vol. 11, no. 4, pp. 713–726, 2018.

[11] M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[12] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011–1026, 2011.
[13] H. Wu, X. Hua, Z. Li, and S. Ren, "Resource and Instance Hour Minimization for Deadline Constrained DAG Applications Using Computer Clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 885–899, 2016.
[14] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.
[15] J. Sahni and P. Vidyarthi, "A Cost-Effective Deadline-Constrained Dynamic Scheduling Algorithm for Scientific Workflows in a Cloud Environment," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 2–18, 2018.
[16] A. Deldari, M. Naghibzadeh, and S. Abrishami, "CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud," *Journal of Supercomputing*, vol. 73, no. 2, pp. 756–781, 2017.
[17] Z. Zhu and X. Tang, "Deadline-constrained workflow scheduling in IaaS clouds with multi-resource packing," *Future Generation Computer Systems*, vol. 101, pp. 880–893, 2019.
[18] 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 *International Conference on Cloud Computing, CloudComp 2009*. Springer, 2010, pp. 115–131.
[19] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, 2013.
[20] M. Niu, B. Cheng, and J. L. Chen, "GMAS: A Geo-Aware MAS-Based Workflow Allocation Approach on Hybrid-Edge-Cloud Environment," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 574–581.
[21] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary Multi-Objective Workflow Scheduling in Cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1344–1357, 2016.
[22] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. Su, and K. Vahi, "Characterization of scientific workflows," in *2008 Third Workshop on Workflows in Support of Large-Scale Science*, 2008, pp. 1–10.
[23] K. Deng, K. Ren, M. Zhu, and J. Song, "A Data and Task Co-Scheduling Algorithm for Scientific Cloud Workflows," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 349–362, 2020.