An Effective Cloud Workflow Scheduling Approach Combining PSO and Idle Time Slot-Aware Rules

Yun Wang and Xingquan Zuo, *Senior Member, IEEE*

***Abstract*—Workflow scheduling is a key issue and remains a challenging problem in cloud computing. Faced with the large number of virtual machine (VM) types offered by cloud providers, cloud users need to choose the most appropriate VM type for each task. Multiple task scheduling sequences exist in a workflow application. Different task scheduling sequences have a significant impact on the scheduling performance. It is not easy to determine the most appropriate set of VM types for tasks and the best task scheduling sequence. Besides, the idle time slots on VM instances should be used fully to increase resources' utilization and save the execution cost of a workflow. This paper considers these three aspects simultaneously and proposes a cloud workflow scheduling approach which combines particle swarm optimization (PSO) and idle time slot-aware rules, to minimize the execution cost of a workflow application under a deadline constraint. A new particle encoding is devised to represent the VM type required by each task and the scheduling sequence of tasks. An idle time slot-aware decoding procedure is proposed to decode a particle into a scheduling solution. To handle tasks' invalid priorities caused by the randomness of PSO, a repair method is used to repair those priorities to produce valid task scheduling sequences. The proposed approach is compared with state-of-the-art cloud workflow scheduling algorithms. Experiments show that the proposed approach outperforms the comparative algorithms in terms of both of the execution cost and the success rate in meeting the deadline.**

***Index Terms*—Cloud computing, idle time slot, particle swarm optimization, task scheduling sequence, workflow scheduling.**

I. Introduction

IN scientific computing communities, such as astronomy, physics, and bioinformatics, there are many large-scale and complex workflow applications consisting of tasks with data dependencies amongst them [1]. Those applications must be deployed in high-performance distributed computing environments for rapid execution. Cloud computing [2] offers cloud users elastic resources that can be acquired and released on demand. Cloud users pay for the leased resources on a pay-as-you-go basis. Such flexible resource provisioning and pay-as-you-go strategy attract enterprises or research institutes to run their workflow applications on clouds at low costs without the need of purchasing and maintaining any infrastructure.

In cloud computing, IT resources are often encapsulated as virtual machines (VMs). The running VMs are called VM instances. Cloud users usually want to obtain the computation result of a workflow within a given deadline at lower execution cost. Generally, the more computing power a VM has, the higher its price. To balance the execution cost and runtime of a workflow, scheduling the tasks of a workflow onto VM instances [3] is very vital for cloud computing. However, the flexible management of cloud resources and the complex workflow structure makes the cloud workflow scheduling challenging.

Cloud providers offer various VM types with different configurations (e.g., CPU, memory and disk size, and price) for users to choose. The number of VM types has been increasing. For example, the number of VM types provided by Amazon elastic computing cloud (Amazon EC2) has been recently increased from 8 to more than 35 [4]. Faced with so many VM types, scheduling algorithms need to choose the most appropriate VM type for each task to achieve the lowest cost for the whole workflow. However, when choosing a VM type for a task, current scheduling algorithms typically only consider the VM type that has the cheapest price and can meet the resources needed to complete the task, while ignoring the impact of the chosen VM type on other subsequent tasks. This manner of choosing VM type may encourage subsequent tasks to be scheduled on faster VM instances, thereby increasing the cost of a workflow [5].

Scheduling algorithms usually assign each task in the order of a task scheduling sequence to a VM instance. The task scheduling sequence has a significant impact on the execution time and cost of a workflow application [6]. For a workflow application, although precedence constraints exist for many tasks, there are many parallel tasks which do not directly have data dependencies with each other. On the premise of keeping the dependencies of tasks, the scheduling order of parallel

Manuscript received October 26, 2020; accepted November 19, 2020. This work was supported in part by the National Natural Science Foundation of China (61874204, 61663028, 61703199), in part by the Science and Technology Plan Project of Jiangxi Provincial Education Department (GJJ190959). Recommended by Associate Editor Shangce Gao. *(Corresponding author: Xingquan Zuo.)*

Citation: Y. Wang and X. Q. Zuo, " An effective cloud workflow scheduling approach combining pso and idle time slot-aware rules," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 5, pp. 1079–1094, May 2021.

Y. Wang is with the School of Computing Science, Beijing University of Posts and Telecommunications, Beijing 100876, and also with the School of Information Engineering, Nanchang Institute of Technology, Nanchang 330099, China (e-mail: wangyun@nit.edu.cn).

X. Q. Zuo is with the School of Computing Science, Beijing University of Posts and Telecommunications, Beijing 100876, and also with the Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education, Beijing 100876, China (e-mail: zuoxq@bupt.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JAS.2021.1003982

tasks can be arbitrary. Thus, a workflow application may have lots of different task scheduling sequences. Moreover, the complex workflow structure makes a workflow have an exponential number of different task scheduling sequences. However, usually only a fixed task scheduling sequence is used in current scheduling algorithms, which restricts the search space for obtaining a better solution. It is necessary to determine the most appropriate task scheduling sequence to achieve a scheduling solution with better performance.

In addition, due to the precedence constraint amongst workflow tasks, idle time slots exist on VM instances. The more idle time slots a VM instance has, the lower its utilization is. Moreover, the billing interval for VM instances is usually set to one hour or one minute by cloud providers, such as Amazon EC2 and Google compute engine cloud. That means even if a VM instance is used for less than one billing period (e.g., one hour), users must pay for the whole hour. If a VM instance has an idle time slot between two running tasks and another task can be run within the slot, scheduling the task to that slot will save cost. It is vital to consider the scheduling of tasks to the idle time slots of VM instances to reduce the workflow's cost and improve VMs' utilization.

In this paper, we propose a new scheduling approach for the deadline constrained cloud workflow scheduling problem, with simultaneous consideration of aforementioned three aspects. This approach, termed HPSO, combines PSO and idle time slots-aware scheduling rules. To be specific, the VM type required by each task and the priority of each task are indicated by a particle in PSO. It means that a particle represents a specific mapping of tasks to VM types and a scheduling sequence of tasks. Idle time slot-aware scheduling rules are proposed to decode a particle into a scheduling solution. Those rules assign each task to a leased VM instance or to a new instance, making full use of idle time slots of VMs.

The contributions of this paper include:

1) A new approach for cloud workflow scheduling is proposed, which combines a meta-heuristic (PSO) and heuristic rules (idle time slot-aware scheduling rules).

2) A particle coding scheme is proposed, which represents the assignment of VM types and scheduling sequences for all tasks in workflow.

3) A PSO is devised to find the best particle, that is, the most appropriate assignment of VM types for tasks and the best task scheduling sequence, to achieve the best scheduling performance. To avoid infeasible solutions during the search of PSO, a repair method is devised to repair tasks' invalid priorities to generate a valid task scheduling sequence.

4) Idle time slot-aware scheduling rules are devised to decode a particle into a scheduling solution. The decoding procedure makes full use of idle time slots of leased VM instances to improve VMs' utilization and minimize costs.

The remainder of this paper is organized as follows. Section II outlines current cloud workflow scheduling algorithms. Section III introduces the model and the basic elements of the scheduling problem. Section IV presents the details of the proposed approach. Experimental results are described and discussed in Section V. Finally, we conclude our work and give insight into future works in Section VI.

II. Related Work

Cloud workflow scheduling is a well-known nondeterministic polynomial (NP)-complete problem [7]. An exact approach cannot find the optimal solution within acceptable computational time for large-scale problem instances. Therefore, current research typically adopts meta-heuristic or heuristic rules to solve this problem. The former uses meta-heuristics, such as PSO, genetic algorithm (GA), and ant colony optimization (ACO) to find a near-optimal solution. The latter obtains an approximate solution to the problem quickly using scheduling rules. Compared with heuristic rules, meta-heuristics can find higher quality solutions as they explore solutions by a guided search but take longer runtime [8].

In current research, many quality of service (QoS) metrics such as cost, makespan [9]–[12], energy consumption [13]–[15], budget [16]–[18], and security [19], [20] are often regarded as optimization objectives or constraints of the problem. This section mainly reviews literature on the deadline constrained cloud workflow scheduling problem, as those are related to our work. Approaches in those literature can be divided into two parts: meta-heuristic based scheduling approaches [21]–[30] and heuristic rules based scheduling approaches [5], [31]–[36].

A. Meta-Heuristic Based Cloud Workflow Scheduling

Rodriguez and Buyya [21] adopted a PSO method (SPSO) to minimize the workflow execution cost while meeting deadline constraint, in which the effect of VM performance variation is considered. SPSO maps tasks onto VM instances by a particle and determines a scheduling solution without considering idle time slots of leased VM instances. The number of VM instances needs to be predetermined. In [22], a GA-based approach was proposed for the cloud workflow scheduling with deadline constraint, where an adaptive penalty function is used for the strict constraints and the coevolution approach is used to adjust the crossover and mutation probabilities. Chen *et al.* [23] used dynamic objective strategy and proposed a dynamic objective GA approach to solve the cloud workflow scheduling with deadline constraint. Chen *et al.* [24] further developed an ACO-based approach to solve the same problem in [23]. Considering the uncertainty of the runtime of tasks, Jia *et al.* [25] designed a new estimation model of the tasks' runtime based on historical data. On this basis, an adaptive ACO based cloud workflow scheduling algorithm was proposed. In addition to PSO, GA and ACO, other meta-heuristics, such as flog leaping [26] and biogeography-based optimizations [27] have been applied to workflow scheduling problems.

In those meta-heuristics, the coding of scheduling solution usually represents the mapping of tasks to a set of VM instances. However, due to the infinite number of VM instances in clouds, a limited number of VM instances must be predetermined. But, it is not easy to predetermine an appropriate set of instances. Moreover, those algorithms usually only use a single task scheduling sequence while

ignoring the impact of different task scheduling sequences on the cloud workflow scheduling performance.

Zuo *et al.* [28] proposed a self-adaptive learning PSO-based scheduling approach by considering multiple different task scheduling sequences. However, the algorithm was used for the scheduling of independent tasks on hybrid cloud, and not for the workflow application. Wu *et al.* [29] proposed a meta-heuristic algorithm (LACO) and employed ACO to carry out deadline-constrained cost optimization. In LACO, different task scheduling sequences are constructed by different ant individuals. But the idle time slots on leased resources are not exploited adequately, and the method of determining the VM type required by each task is different from ours.

In addition, Yuan *et al.* [30] considered the uncertainty of arriving tasks and deemed that the energy price of private cloud data center (CDC) and the execution price of public clouds have the temporal diversity. A hybrid scheduling algorithm combining PSO and simulated annealing method (TTSA) is proposed to solve them. TTSA is used to schedule parallel tasks; not workflow tasks with data dependencies. Moreover, TTSA is to schedule all parallel tasks from a private CDC perspective, and our work is to schedule one workflow application from the cloud users' perspective. Finally, the optimization objective of TTSA is to minimize the cost for private CDC, including the energy cost of private CDC and the execution cost generated by outsourcing tasks to public clouds, whereas ours is to minimize the cost of executing a workflow on a public cloud.

B. Heuristic Rules Based Cloud Workflow Scheduling

Heterogeneous earliest finish time (HEFT) [31] is one of the most widely used heuristic scheduling rules. It is first used to construct a task scheduling sequence based on tasks' priorities, and then assigns each task in order of the scheduling sequence to a processor that can meet the users' needs, such as the earliest finish time. Although HEFT was originally proposed for workflow scheduling in a limited number of heterogeneous processors, many modified versions of HEFT have been proposed for cloud workflow scheduling [16], [37], [38].

Abrishami *et al.* [32] introduced the concept of partial critical path (PCP) and proposed two algorithms, which are called IaaS cloud partial critical paths (ICPCP) and IaaS cloud partial critical paths with deadline distribution (ICPCPD2), respectively. ICPCP tries to minimize the execution cost of a workflow by scheduling a PCP path on a cheapest VM instance, which can finish all the tasks of the PCP before their latest finish time. Sahni and Vidgarthi [5] proposed a just-in-time scheduling algorithm (JIT) while taking into account the VM performance variability and instance acquisition. Arabnejad *et al.* [33] proposed two algorithms, proportional deadline constrained (PDC) and deadline constrained critical path (DCCP), for the deadline-based workflow scheduling on dynamically provisioned cloud resources. In the aforementioned algorithms, the VM type for each task is usually chosen by heuristic information (such as the cheapest VM). Moreover, only the remaining time of the last time interval of leased instances is typically used while ignoring the availability of other idle time slots. Our approach selects the best VM type for each task by a PSO and considers the effective use of all idle time slots.

Recently, some studies [34]–[36] considered the efficient use of idle time slots to improve the performance of workflow scheduling. To mitigate the effect of performance variation of resources on soft deadlines of workflow applications, Calheiros and Buyya [34] used the idle time slots of leased resources and budget surplus to replicate task so as to meet the deadline constraint. The minimization of the execution cost of a workflow is not considered in their work. By considering realistic factors such as software setup time and data transfer time, Li and Cai [35] proposed a multi-rules based heuristic to solve the deadline-based workflow scheduling. Furthermore, three priority rules are developed to allocate tasks to appropriate available time slots. The heuristics for scheduling workflow tasks are based on reserved resources in clouds. How to determine the best task scheduling sequence was not considered.

To our knowledge, our work is the first study considering the following three aspects simultaneously: selecting the most appropriate VM type for each task, determining the best task scheduling sequence, and effectively using idle time slots. A PSO combined with idle time slot-aware rules is proposed to solve those aspects, to minimize the execution cost of a workflow while meeting the deadline constraint.

III. Cloud Workflow Scheduling Problem and Formulation

A. Workflow Model

A workflow is usually represented by a directed acyclic graph (DAG) consisting of vertexes and edges [39]. A DAG is formulated as a tuple $G = <T, E>$, where $T = \{t_1, t_2, ..., t_n\}$ is a set of vertices corresponding to tasks of a workflow, n denotes the number of tasks, and $E = \{e_{ij} | t_i, t_j \in T\}$ is a set of directed edges reflecting data dependencies amongst tasks. For instance, an edge e_{ij} means that there is the precedence constraint between t_i and t_j, that is, t_i is the direct predecessor (parent) of t_j, and t_j is the direct successor (child) of t_i. Every edge e_{ij} has a weight to represent the size of data transferred from t_i to t_j. A task may have one or more parents or children, and the task cannot be executed until all its parents have been executed and all input data required by the task has been received. All direct predecessors and successors of t_i are defined respectively as follows:

$$\rho(t_i) = \{t_j | e_{ji} \in E \quad \text{and} \quad t_i, t_j \in T\} \tag{1}$$

$$s(t_i) = \{t_j | e_{ij} \in E \quad \text{and} \quad t_i, t_j \in T\}. \tag{2}$$

In DAG, a task without any parent is an entry task t_{entry}. Similarly, a task without any children is called an exit task t_{exit}. There may be multiple entry tasks and exit tasks in a DAG. Fig.1 shows an example workflow with 5 tasks, where t_1 is the entry task and t_5 is the exit task.

B. Cloud Resource Model

The cloud model consists of a single data center and has various VM types. A set $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ represents all

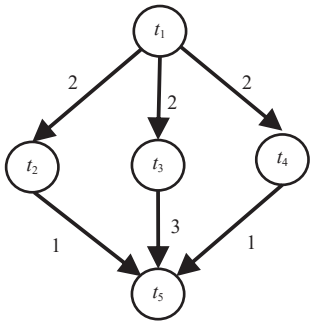

Fig. 1. An example workflow with 5 tasks.

VM types, and m is the number of VM types. In this paper, all VM types are assumed to have enough memory to execute a workflow and only the CPU and network bandwidth of each VM type are considered. The number of instances a user can rent from the cloud is unlimited.

This paper assumes that the CPU capacity represented by floating-point operation per second (FLOPS) is available either from a cloud provider or estimated by a performance estimation method [40]. The workload of a task t_i is assumed to be known in advance. Therefore, the execution time of task t_i on VM instance λ_j, $\chi(t_i, \lambda_j)$, is defined as follows:

$$\chi(t_i,\lambda_j)=\frac{w(t_i)}{\alpha(\lambda_j)} \tag{3}$$

where $\alpha(\lambda_j)$ represents the CPU capacity of λ_j, and $w(t_i)$ is the workload of t_i.

All VM instances are in the same data center. The average network bandwidth β between instances is roughly the same. The data transfer time between t_i and t_j, $\gamma(t_i, t_j)$, is defined as (4), where the amount of data transferred between the two tasks, $\mu(t_i, t_j)$, is known. Note that the transfer time between two tasks is 0 when they are executed on the same VM instance.

$$\gamma(t_i,t_j)=\frac{\mu(t_i,t_j)}{\beta}. \tag{4}$$

According to the pay-per-use strategy, all leased instances are charged for the number of used billing intervals. This paper assumes that the size of billing interval is one hour, just like the on-demand VM instances of Amazon EC2. The cost of data transfer is not considered because many commercial cloud providers do not charge the cost within a data center. In addition, we assume that the tasks of a workflow application cannot be preempted.

C. Basic Conceptions

This section introduces some basic concepts of workflow that are used in Section IV-D. The earliest start time of task t_i on one instance λ_j, $EST(t_i, \lambda_j)$, is determined by the finish time of all parents of t_i. It means that all parents of t_i must be completed before executing t_i. Besides, if t_i is an entry task, its earliest start time on λ_j is 0. $EST(t_i, \lambda_j)$ is computed as follows:

$$\begin{cases} EST(t_{entry},\lambda_j)=0 \\ EST(t_i,\lambda_j)=\max\limits_{t_a\in\rho(t_i)}\{F(t_a,\lambda_k)+\gamma(t_a,t_i)\} \end{cases} \tag{5}$$

where $F(t_a, \lambda_k)$ is the actual finish time of t_a on instance λ_k, and t_a is a parent of t_i and has been scheduled on λ_k. In order to utilize the idle time slots of instance λ_j, the calculation of $EST(t_i, \lambda_j)$ does not consider the final available time of λ_j in this paper. The actual start time of t_i on instance λ_j, $S(t_i, \lambda_j)$, must be greater than or equal its earliest start time.

The actual finish time of t_i, $F(t_i, \lambda_j)$, equals the sum of $S(t_i, \lambda_j)$ and $\chi(t_i, \lambda_j)$, and is computed as follows:

$$F(t_i,\lambda_j)=S(t_i,\lambda_j)+\chi(t_i,\lambda_j). \tag{6}$$

The latest finish time of t_i, $LFT(t_i)$, is the latest time that t_i should be finished to ensure a workflow will be finished before its deadline Δ.

$$\begin{cases} LFT(t_{exit})=\Delta \\ LFT(t_i)=\min\limits_{t_c\in s(t_i)}\{LFT(t_c)-MET(t_c)-\gamma(t_i,t_c)\} \end{cases} \tag{7}$$

where $MET(t_c)$ is the execution time of t_c based on the fastest VM, and t_c is a child task of t_i.

For more details about the above basic concepts of workflow, please refer to literature [31] and [39].

D. Cloud Scheduling Problem Description

This paper focuses on minimizing the cost of executing a workflow in clouds while meeting a deadline constraint. A scheduling solution is expressed as $S = (R, M, \Theta)$. $R = \{\lambda_1, \lambda_2, \ldots, \lambda_l\}$ is a set of l leased VM instances that will be used to execute tasks. Each VM instance in R has four attributes, that is, $\lambda_i = (\pi_i, B_i, Z_i, L_i)$, representing the VM type of VM instance λ_i, the lease start time and end time of λ_i, and tasks list scheduled on the instance, respectively. $M = \{m_1, m_2, \ldots, m_n\}$ represents the mapping of tasks to instances in R. $m_i = (\lambda_j, S(t_i), F(t_i))$ means that task t_i is scheduled on the instance λ_j and is expected to start executing t_i at time $S(t_i)$, and finish by time $F(t_i)$. The size of M is equal to the number of tasks n in the workflow, and the size of R is less than or equal to n. Θ is a task scheduling sequence and must follow the dependencies amongst tasks.

Based on $S = (R, M, \Theta)$, the execution cost Ψ and the execution time Ω of a workflow are calculated by the following formulas:

$$\Psi=\sum_{i=1}^{l}\mu_{\pi_i}\times\left\lceil\frac{(Z_i-B_i)}{\tau}\right\rceil \tag{8}$$

$$\Omega=\max\{F(t_i),t_i\in T\}. \tag{9}$$

In (8), $\lceil (Z_i - B_i)/\tau \rceil$ is the number of the billing intervals of instance λ_i used by a workflow, μ_{π_i} is the unit price of the VM type π_i, and τ is the length of the billing interval. Ω in (9) is also called the makespan. Thus, the cloud scheduling problem studied in this paper is to optimize the following objective:

$$\begin{aligned} &\text{minimize } \Psi \\ &\text{s. t. } \quad \Omega\le\Delta. \end{aligned} \tag{10}$$

where Δ is the deadline of a workflow.

IV. Proposed Approach

PSO is a stochastic optimization algorithm which is inspired by the foraging behavior of bird flocks [41]. Because PSO has few parameters and is easy to implement, it has been widely

applied to numerous fields, including production scheduling problems [42] and cloud resources scheduling [43], [44]. This paper uses PSO combined with idle time slot-aware rules to solve the workflow scheduling on clouds.

A. PSO for Cloud Workflow Scheduling

PSO has a population of particles and each particle represents a possible solution to the problem. The population size, N, refers to the number of particles in the population. The search for the best solution is guided by a fitness function which evaluates the quality of each particle. Each particle consists of a position and velocity. The velocity of the ith particle at the tth iteration is denoted as $v_i^t = (v_{i1}^t, v_{i2}^t, ..., v_{iD}^t)$, and its position is $x_i^t = (x_{i1}^t, x_{i2}^t, ..., x_{iD}^t)$, where D is the dimensions of search space. The historical best position $p_i^t = (p_{i1}^t, p_{i2}^t, ..., p_{iD}^t)$ represents the best previous position yielding the best fitness value for the i th particle at the tth iteration. The global best particle $g^t = (g_1^t, g_2^t, ..., g_D^t)$ is the global best position found by all particles so far. At each of iterations of PSO, the velocity and position of the dth dimension of each particle is updated using the following formulas:

$$v_{id}^{t+1} = \omega v_{id}^t + c_1 r_1 (p_{id}^t - x_{id}^t) + c_2 r_2 (g_d^t - x_{id}^t) \quad (11)$$

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1} \quad (12)$$

where inertia weight ω is used to balance the global and local search; r_1 and r_2 are uniform random numbers in [0, 1]; c_1 and c_2 are acceleration factors to control the influence of the social and cognitive components, and their values usually are set as 2 [41].

A pseudocode of PSO for the cloud workflow scheduling problem is described in Algorithm 1. It evolves iteratively a population of particles until the number of iterations is reached (lines 5–15) and then the scheduling solution of the global best particle is regarded as the final scheduling solution (line 16).

There are several issues to consider when using the PSO to solve the scheduling problem: the first is the encoding representation of a particle, which is elaborated in Section IV-B; the second is how to decode a particle into a scheduling solution (decoding procedure), which is shown in line 9 and detailed in Section IV-D; the third is to evaluate a particle by a fitness function, which reflects the optimization objectives. In this paper, the execution cost Ψ and the execution time Ω are calculated in the fitness function (in Section IV-E), and Ψ is used to evaluate a particle (line 10). The fourth issue is the population initialization that is described in Section IV-F; and the final issue is the treatment of the deadline constrain of the scheduling problem.

During the search of the PSO, some infeasible solutions that violate the deadline constraint may be generated. To handle those infeasible solutions, the constraint handing technique in [45] is introduced into the PSO. A solution is infeasible if it does not satisfy the deadline constraint. There are three cases for two solutions' comparison: the solution with a better fitness value (execution cost) is better if the two solutions are both feasible; the feasible one is better if one solution is feasible and the other is unfeasible; the solution with the smaller execution time is better if both solutions are unfeasible. In lines 11 and 13, the historical best position of a particle and the global best particle are updated under the constraint-handing technique.

In addition, in line 1, the workflow is preprocessed to merge the " pipeline pair" tasks into a single task, which helps to reduce the runtime overhead of the scheduling algorithm and save the data transfer time between tasks [10]. The "pipeline pair" tasks refer to a special pair of tasks t_i and t_j which have a one to one relationship. That is, t_i only has one child task t_j and t_j only has one parent task t_i.

Algorithm 1 PSO for Cloud Workflow Scheduling

Input: The DAG of a workflow $G = <T, E>$.
Output: The scheduling solution $S = (R, M, \Theta)$.
1: Preprocessing the workflow;
2: Set parameters, such as population size N and inertia weight;
3: Initialize the population;
4: Let the number of iterations, $t = 1$;
5: While $t \leq$ the total number of iterations do
6: For each particle q_i, $i \in \{1, 2, \ldots, N\}$ do
7: Update the velocity v_i^t by (11);
8: Update the position x_i^t by (12);
9: Decode q_i to a solution S_i by Algorithm 3;
10: According to S_i, calculate Ψ and Ω by Algorithm 4;
11: Update the historical best position p_i of q_i;
12: End for
13: Update the global best particle g;
14: $t = t+1$;
15: End while
16: Output the scheduling solution S of g;

B. Encoding Scheme

A key issue for PSO to solve a problem is to devise an effective particle encoding. Although VM instances in clouds are infinite, cloud providers provide a finite number of VM types. Inspired by this, the particle encoding only needs to consider the mapping of tasks to VM types (instead of VM instances). Besides, the priority of each task, which determines its scheduling order in all tasks, is encoded in the particle. A task scheduling sequence is obtained by sorting tasks' priorities in ascending order. Therefore, a particle contains two parts: the first part (dimensions from 1 to n) embodies the mapping of tasks to VM types, and the second part (dimensions from $n+1$ to $2n$) determines the priority for each task. The dimension of a particle D is equal to twice the number of workflow tasks. The value of each dimension in the first part falls within the real range of [1, m]. The values of the second part can be any positive real number.

Because PSO is for continuous optimization problems, whereas the workflow scheduling problem is a combinatorial one, the value of each dimension must be rounded to the nearest integer. Specifically, in the first part of a particle, dimensions from 1 to n correspond to tasks t_1 to t_n, respectively. The rounded nearest integer of each dimension represents an index of VM type, which means the

corresponding task is assigned to an instance with the same VM type. In the second part of a particle, the rounded nearest integer of dimension $i \in [n+1, 2n]$ represents the priority of task t_{i-n}.

An example of an encoded particle for the workflow in Fig. 1 is shown in Fig. 2, where the number of VM types is assumed to be 4. In this example, dimension 1 corresponds to task t_1 and its value is 2.1. This means that the VM type required by t_1 is π_2. The values of dimensions 2–5 follow the same logic. The values of dimensions 6–10 are rounded to 2, 9, 11, 3, and 14, indicating the priorities of t_1–t_5, respectively. Sorting those values in ascending order, the scheduling sequence of tasks is: t_1-> t_4-> t_2-> t_3-> t_5.

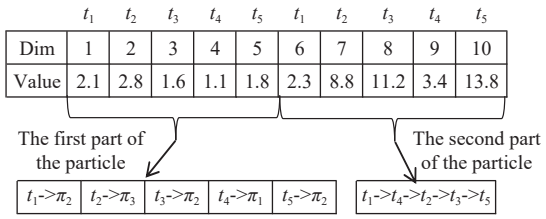

Fig. 2. A particle encoding for the workflow in Fig. 1.

By this encoding, different mapping combinations of tasks to VM types and different task scheduling sequences are represented by different particles. On this basis, an idle time slot-aware heuristic decoding procedure in Section IV-D can identify the VM instances to be leased and the scheduling of all tasks on those leased instances.

C. Repairing Method for Invalid Tasks's Priorities

A feasible task scheduling sequence must satisfy the data dependencies amongst tasks. However, those cannot be guaranteed during the iterative evolution because of the stochastic feature of PSOs. This paper sorts priorities of all tasks in ascending order to obtain a task scheduling sequence. This means the priority of each task must be greater than those of all its parents and cannot be the same as each other. Otherwise, the priority of the task is invalid.

A simple repairing method is designed to repair tasks' invalid priorities. Its main idea is to modify the invalid priority of a task to the maximum value of the priorities of all its parents plus 1, while ensuring that the modified priority is not equal to any other tasks' priorities.

Algorithm 2 gives the detailed steps of the repairing procedure, where T is a set of workflow tasks, and set V is used to store those tasks with valid priorities and is initially set to be empty (line 2). The priority of an entry task t_{entry} with the minimum priority is deemed as valid (line 3). Based on this priority, lines 5–18 gradually judge whether the priorities of other tasks are valid. First, those tasks whose parents are in set V are selected from T and represented by set Q (line 6,). Then the following steps are to judge whether the priority of each task t_j in Q is valid one by one (lines 8–17). If the priority of t_j, $\eta(j)$, is not bigger than the maximum priority value of all its parents, $p^{\max}$, it is modified to $p^{\max}+1$ (lines 10–12). Moreover, if the priority of t_j is identical to that of one task in set V, it is modified to $v^{\max}+1$ (lines 13–15), where $v^{\max}$ is the maximum of the priorities of all tasks in set V (line 7).

Through this, the priority of each task is bigger than those of all its parents and all tasks have different priorities. Finally, once a task has a valid priority or the invalid priority of the task has been modified to a valid one, the task is added into V and removed from T (line 16).

Algorithm 2 Repair Invalid Priorities of Tasks.

Input: A particle q and a set of tasks $T=\{t_1, t_2, \ldots, t_n\}$.
Output: The repaired particle.
1: $\eta(1{:}n)$ = round ($q(n+1{:}2n)$);
2: $V=\emptyset$;
3: Find an entry task t_{entry} with the minimum priority;
4: $V = V \cup \{t_{entry}\}$, and $T = T \backslash \{t_{entry}\}$;
5: While (T is not empty)
6: Find all tasks whose parents are in V, denoted by Q;
7: $v^{\max}$ = max(priorities of tasks in V);
8: For (each task t_j in Q)
9: $p^{\max}$ = max(priorities of t_j's parents);
10: If $\eta(j) \le p^{\max}$
11: $\eta(j) = p^{\max}+1$;
12: End if
13: If $\eta(j)$ = = the priority of one task in V
14: $\eta(j) = v^{\max}+1$;
15: End if
16: $V = V \cup \{t_j\}$, and $T = T \backslash \{t_j\}$;
17: End for
18: End while
19: $q(n+1{:}2n) = \eta(1{:}n)$;
20: Output the repaired particle;

D. Idle Time Slot-Aware Decoding Procedure

An idle time slot-aware decoding procedure is proposed to decode a particle into a scheduling solution. The decoding procedure makes full use of the idle time slots of leased VM instances to improve resource utilization and decrease execution cost of a workflow. Meanwhile, it can schedule tasks under their precedence constraints and tends to schedule them on the same VM instance to save the data transfer time between them.

Algorithm 3 presents the detailed steps. In line 1, R, M, and H are set as empty. Set H is used to store the applicable instances of a task t_j. Herein, if a leased instance has one or more idle time slots that can be used to finish t_j before t_j's latest finish time, $LFT(t_j)$, it is termed an applicable instance of t_j. $LFT(t_j)$ is calculated by (7). Each task is scheduled in order of the scheduling sequence Θ to a leased instance in R or a new instance (lines 4–17). In line 5, the VM type π_j, required by t_j is obtained from the jth dimension of the particle. Then, serial and parallel instances with the same type π_j, denoted by X and Y, respectively, are selected from R (line 6). For t_j, serial instances refer to those instances that have one or more parents of t_j scheduled, and parallel instances are those that do not schedule any one of t_j's parents. Note that the VM type of an instance on which a task is scheduled must be the same as that required by the task.

In order to save the data transfer time, those serial instances

which are applicable to t_j are first selected from X and recorded by H (line 7). If there is no applicable instance of t_j in X, namely the set H is empty (line 8), continue to find parallel instances that applicable to t_j in Y (line 9). Finally, if there is still no applicable instance can be used to schedule t_j in Y (line 13), a new instance λ_n with π_j is launched for t_j (line 14).

There may be multiple applicable instances in H. The instance λ_a, which has the smallest cost difference between its execution cost before scheduling t_j and that after scheduling t_j, is chosen to execute t_j from H (lines 11–12). The execution cost of an instance is calculated by (8), where $l = 1$. Furthermore, if there are multiple instances with the same smallest cost difference in H, select one of those instances randomly for t_j.

Algorithm 3 An Idle Time Slot-Aware Decoding Procedure.

Input: A particle q (1:2n).
Output: A scheduling solution $S = (R, M, \Theta)$.
1: $R = \emptyset$, $M = \emptyset$, $H = \emptyset$;
2: Repair tasks' invalid priorities in the particle q by Algorithm 2;
3: Get a feasible scheduling sequence Θ based on the particle q;
4: For each task $t_j, j \in \{1, 2, \ldots, n\}$ in Θ do
5: $\pi_j = q(j)$;
6: Select serial and parallel instances with VM type π_j from R, denoted as X and Y, respectively;
7: Search the applicable instances of t_j from X, and recorded them by H;
8: If H is empty then
9: Continue to search the applicable instances of t_j (also recorded by H) from Y;
10: End if
11: If H is not empty then
12: Allocate t_j to the instance $\lambda_a \in H$ with the smallest cost difference;
13: Else
14: Launch a new instance λ_n with π_j for t_j;
15: End if
16: Update R and M;
17: End for
18: Output $S = (R, M, \Theta)$.

All applicable instances of t_j in X and Y (lines 7 and 9) are identified by an insertion-based policy. The length of an idle time slot is equal to the difference between the execution start time and finish time of two tasks that were consecutively scheduled on the same instance [31]. If an idle time slot between t_x and t_k is available to t_j, it means that $S(t_j) \geq F(t_x)$, $F(t_j) \leq S(t_k)$, and $F(t_j) \leq LFT(t_j)$. Note that scheduling a task on an idle time slot must satisfy its precedence constraints. Therefore, the insertion-based policy first calculates the $EST(t_j)$ of t_j on an instance by (5), and then check all idle time slots on the instance to determine whether available time slots exists to execute t_j.

There may be multiple idle time slots available for executing t_j on an applicable instance, and t_j is scheduled in the first time slot in chronological order. The $S(t_j)$ on an applicable instance is finally determined during checking the idle time slots by the insertion-based policy, and $F(t_j)$ is calculated by (6).

E. Fitness Function

In this paper, the total execution cost of a scheduling solution $S_i = (R, M, \Theta)$ derived from a particle is used as the fitness value of the particle. The total execution time of S_i is used to judge whether the particle is feasible. For example, if the total execution cost of a particle q_1 is less than that of another particle q_2, it means q_1 is better than q_2. If the total execution time of a particle is bigger than the deadline, the particle is unfeasible; otherwise, it is feasible.

Algorithm 4 Fitness Function.

Input: A solution $S_i = (R, M, \Theta)$ of a particle
Output: The execution cost and time of a workflow, Ψ and Ω.
1: Ψ is calculated by (8), based on the R of S_i;
2: Ω is computed by (9), according to the M of S_i;
3: Output Ψ and Ω;

In Algorithm 4, all leased instances are in R, and the mappings between tasks and leased instances are in M. The lease expenses of all leased instances are summed up as the total execution cost by (8), and the total execution time is computed by (9). Finally, output the total execution cost and time of $S_i = (R, M, \Theta)$ (line 3).

F. Population Initialization

In order to generate N particles with valid task scheduling sequences, the upward and downward ranks of a task, which are usually used in the list-based scheduling algorithms, are adopted to compute tasks' priority in the initial population.

The upward rank of a task t_i is the critical path length from t_i to the exit task, including t_i' s average execution time. The down rank of t_i is the longest distance from the entry task to it, excluding the average execution time of t_i. Please refer to [39] for the calculations of tasks' upward and downward ranks.

In the initiation of population, two populations, O_1 and O_2, are first generated randomly, and each population has N particles. For each particle in O_1, the value of dimension $i \in [n+1, 2n]$ is set to the downward rank of task t_{i-n}. Besides, because this paper obtains the task scheduling sequence by sorting tasks' priorities in ascending order, the priority of each task in O_2 is set as $\sigma - r_u(t_i)$, where σ represents the maximum value of all tasks' upward ranks and $r_u(t_i)$ is the upward rank of t_i. Finally, N particles are selected as the initial population from O_1 and O_2 in ascending order of their fitness values.

V. Performance Evaluation

A. Experimental Settings

Pegasus project [46] publishes some synthetic workflows resembling those used by real world scientific applications, including Montage, Epigenomics, Sipht, CyberShake, and LIGO's Inspiral Analysis (Inspiral). These workflows have different characteristics and have been widely used to evaluate workflow scheduling algorithms. The DAGs of the five workflows are illustrated in Fig. 3. Each workflow has four different task sizes: small (about 30 tasks), medium (about 50

tasks), large (about 100 tasks), and extra-large (about 1000 tasks). The details of each workflow, including its DAG structure, the size of data transferring amongst tasks, and the running time of each task, are stored in an XML file. The corresponding XML files are available from the web site[1].

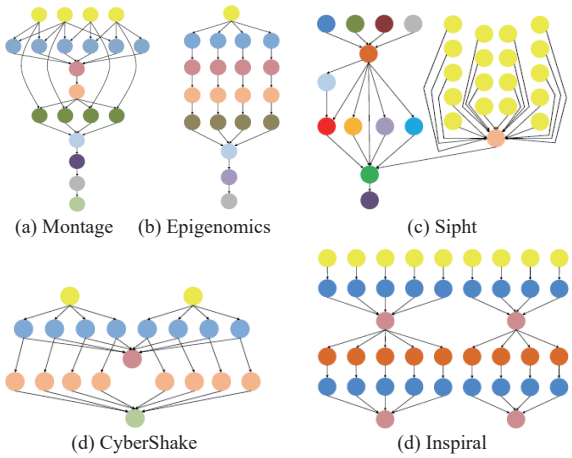

Fig. 3. DAGs of five synthetic workflows.

We use these five workflows to evaluate the HPSO algorithm. For each workflow, eight different deadlines are used to evaluate the capability of HPSO to meet the deadline constraint. These deadlines are calculated based on the fastest and the slowest execution time of a workflow. The slowest execution time, ζ, is the execution time of a workflow whose tasks are all scheduled on the same VM instance with the cheapest VM type. The fastest execution time of a workflow, δ, is estimated by letting each task of the workflow be scheduled separately on a different VM instance with the fastest VM type. The data transmission time is ignored when computing the two time values ζ and δ.

Moreover, the slowest execution time of a workflow is usually two or even more orders of magnitude higher than the fastest one, and the difference between them depends on the structure of the workflow [25]. In order to reflect the capability of different algorithms to meet deadlines, an appropriate set of deadlines must be determined. By trial and error, eight deadlines of a workflow are computed by the following formula:

$$d_i = \begin{cases} \delta + (\dfrac{\zeta - 13\times\delta}{96})\times i \\ \quad \text{for Epigenomics, Sipht, or Inspiral} \\ \delta + (\dfrac{\zeta - 5\times\delta}{32})\times i \\ \quad \text{for Montage or CyberShake} \end{cases} \tag{13}$$

where the values of i are set as 1, 2, …, 8 to calculate eight different deadlines with d_1 being the tightest deadline and d_8 being the loosest one.

A cloud computing environment was simulated using CloudSim toolkit [47] to compare the performance of HPSO and other methods. This paper selects ten VM types with known number of computing unit (ECU) from Amazon EC2, as shown in Table I. Reference [40] shows that an ECU is roughly 4400 million floating point operations per second (MFLOPS). All VM instances are assumed to be in the same data center. The network bandwidth amongst VM instances is roughly equal and is set to 20 Mbps [32]. The billing period of all VM types is assumed to be one hour.

TABLE I
VM TYPES BASED ON AMAZON EC2

VM types	ECUs	Processing capacity (MFLOPS)	Price per hour ($/hour)
m3.medium	3	13 200	0.07
m3.large	6.5	28 600	0.14
m3.xlarge	13	57 200	0.28
m3.2xlarge	26	114 400	0.56
c3.large	7	30 800	0.105
c3.xlarge	14	61 600	0.210
c3.2xlarge	28	123 200	0.42
c3.4xlarge	55	242 000	0.84
r3.large	13	57 200	0.35
r3.4xlarge	52	228 800	1.40

B. Compared Algorithms

To verify the performance of HPSO, five state-of-the-art cloud workflow scheduling algorithms, namely PSO-based scheduling algorithm (SPSO) [21], GA-based scheduling algorithm (EMS-C) [9], ACO-based scheduling algorithm (LACO) [29], and two heuristic scheduling algorithms ICPCP [32] and JIT [5], are selected as comparative algorithms. In current literature on cloud workflow scheduling problems, those algorithms are often used as comparative algorithms. The scheduling problem solved by SPSO, LACO, ICPCP, and JIT is exactly the same as ours (minimizing the execution cost of a workflow while meeting its deadline constraint). EMS-C is a multi-objective optimization algorithm for minimizing the execution time and cost of a workflow simultaneously.

SPSO adopts the PSO to solve the deadline constraint and cost minimization problem, while considering fundamental features of the dynamic provisioning and heterogeneity of unlimited computing resources as well as VM performance variation. It maps tasks onto VM instances by a particle and determines a scheduling solution without considering idle time slots of leased VM instances. The number of VM instances needs to be predetermined.

ICPCP uses the concept of partial critical path (PCP). A PCP contains many tasks and is first scheduled on an already leased instance that can meet the latest finish time of the tasks. If no such instance exists for the path, a new instance with the cheapest VM type able to finish the tasks before their latest finish time is leased for the PCP.

JIT aims to exploit the advantage offered by cloud computing while considering the virtual machine (VM) performance variability and instance acquisition delay to identify a just-in-time schedule of a deadline constrained scientific workflow at lesser costs.

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

LACO employs ant colony optimization to carry out deadline-constrained cost optimization: the ant constructs an ordered task list according to the pheromone trail and probabilistic upward rank. The reason to choose LACO for comparison is that it considers different task scheduling sequences by using different ants and uses a simple heuristic method to decode an ant to a scheduling solution. However, LACO does not consider the selection of the best VM type for each task and the effective utilization of idle time slots of VM instances.

The reason to choose EMS-C for comparison is that it is based on GA. In EMS-C, the encoding of a chromosome includes the mapping of tasks to VM instances, the mapping of VM instances to VM types, and the scheduling order of each task. Because EMS-C is for multi-objective workflow scheduling, it is modified to fit the scheduling problem in this paper. The modified EMS-C (MEMS for short) keeps the single objective of minimizing the execution cost and adds deadline constraint. Like HPSO, MEMS uses the constraint handing technique in [45] to deal with infeasible solutions. Other operations (e.g., crossover and mutation) of MEMS are the same as those of original EMS-C.

C. Algorithm Parameter Discussions

HPSO does not contain any new parameter, except the parameters of PSO (inertia weight ω, acceleration factors c_1 and c_2, population size N, and number of fitness function evaluations K). Those parameters are determined by the following two groups of experiments.

The first group experiment is to determine ω, c_1 and c_2. In order to investigate how the three parameters affect the performance of HPSO, three different values are set for each parameter. For ω, its values are respectively set to 0.5, decreased linearly from 0.9 to 0.4 (0.9~0.4), and decreased linearly from 0.1 to 0.01 (0.1~0.01). The values of c_1 and c_2 are set to (2.0, 2.0), (2.0~0, 0~2.0), and (1.0~0, 0~1.0), respectively. Here, (2.0, 2.0) represents that the values of c_1 and c_2 are all set to be 2.0. (2.0~0, 0~2.0) denotes that c_1 decreases linearly from 2.0 to 0, whereas c_2 increases linearly from 0 to 2.0. The same logic is true for (1.0~0, 0~1.0). Table II shows the average execution costs of Montage-100 under the tightest deadline d_1 for different values of ω, c_1 and c_2. Note that the experiment assumes that N is 20 and K is 1000, which are used to determine ω, c_1 and c_2. In Table II, \$2.40 is the lowest execution cost for Montage-100 under deadline d_1. Therefore, parameters ω, c_1 and c_2 of HPSO are set as (0.1~0.01), (2.0~0), and (0~2.0), respectively.

The second group experiment is to determine the appropriate N and K, i.e., population size and the number of fitness function evaluations, to make a fair comparison of HPSO, LACO, SPSO, and MEMS. Table III shows the average execution costs of Montage-100 under deadline d_1 for different combinations of N and K.

Table III shows that with the increase of K and N, the execution cost values obtained by HPSO (SPSO, MEMS) are reduced from 2.4 (16.90, 23.02) to 1.79 (4.70, 11.79), and the cost values obtained by LACO fall within the range of [2.5. 2.8]. Obviously, along with the increase of N and K, the execution cost of SPSO and MEMS is improved, while the cost of HPSO and LACO does not change much.

TABLE II
THE EXECUTION COSTS OF MONTAGE-100 FOR DIFFERENT INERTIA WEIGHTS AND ACCELERATION FACTORS

ω	(c_1, c_2)		
	(2.0, 2.0)	(2.0~0, 0~2.0)	(1.0~0, 0~1.0)
0.5	2.49	2.65	2.42
0.9~0.4	2.56	2.56	2.63
0.1~0.01	2.70	**2.40**	2.42

TABLE III
THE EXECUTION COSTS OF THE MONTAGE-100 FOR DIFFERENT POPULATION SIZES AND NUMBER OF EVALUATIONS

Size (N)	Algorithms	Number of fitness function evaluations (K)					
		1000	2000	4000	6000	8000	10000
20	HPSO	2.40	2.37	2.28	2.14	1.91	2.14
	LACO	2.84	2.75	2.66	2.71	2.62	2.65
	SPSO	16.90	11.73	9.55	8.20	6.92	6.05
	MEMS	21.69	18.25	17.49	13.07	14.92	13.03
50	HPSO	2.05	2.15	1.94	1.93	1.93	1.94
	LACO	2.73	2.67	2.67	2.60	2.51	2.58
	SPSO	11.28	9.19	8.34	6.65	5.75	5.04
	MEMS	23.02	20.00	16.45	12.68	14.40	15.66
100	HPSO	2.04	2.03	1.93	1.95	1.89	1.79
	LACO	2.69	2.65	2.56	2.58	2.51	2.57
	SPSO	11.12	8.31	6.42	5.87	5.02	4.70
	MEMS	22.80	20.62	17.32	14.07	11.79	14.40

If N and K are set too large, the four meta-heuristic based algorithms will consume too much time for extra-large and large workflows. Thus, the N value of the four algorithms is all set to 20. The K value of SPSO and MEMS is set to 10 000. As HPSO and LACO converge for almost all workflows when K is 1000 (see the third subsection of Section V-D), the K value of HPSO and LACO is all set to be 1000.

The SPSO [21], MEMS [9], and LACO [29] are all designed specifically for the cloud workflow scheduling problem, and in literature [9], [21] and [29], appropriate parameters are suggested. Therefore, other parameters of SPSO, MEMS, and LACO are taken from their literature: for SPSO, c_1, c_2, ω are set to 2, 2, 0.5, respectively; for MEMS, the probabilities of crossover and mutation are given by 1.0 and 1.0/n, respectively, and n is the number of workflow tasks; for LACO, the pheromone and the heuristic information are set to 1 and 2, respectively, and the pheromone evaporation coefficient is 0.2.

HPSO, SPSO, MEMS, and LACO perform 10 independent runs for each workflow application.

D. Results and Discussions

1) Deadline Constraint Evaluation

For a workflow with a deadline d_i, if the average execution time of the workflow scheduled by an algorithm is less than or

TABLE IV
THE SUCCESS RATES OF SIX ALGORITHMS FOR ALL WORKFLOWS

Workflows	HPSO	SPSO	ICPCP	MEMS	LACO	JIT	Workflows	HPSO	SPSO	ICPCP	MEMS	LACO	JIT
CyberShake-1000	100%	100%	100%	100%	100%	100%	CyberShake-100	100%	87.5%	87.5%	75%	87.5%	62.5%
Montage-1000	100%	100%	100%	75%	100%	100%	Montage-100	100%	100%	100%	87.5%	100%	100%
Inspiral-1000	100%	100%	100%	12.5%	100%	100%	Inspiral-100	100%	75%	100%	37.5%	100%	100%
Sipht-1000	100%	100%	100%	100%	87.5%	100%	Sipht-100	100%	100%	100%	87.5%	100%	100%
Epigenomics-997	100%	100%	100%	100%	100%	100%	Epigenomics-100	100%	75%	100%	100%	100%	100%
CyberShake-50	100%	75%	87.5%	50%	100%	62.5%	CyberShake-30	100%	62.5%	50%	0	100%	0
Montage-50	100%	100%	87.5%	87.5%	100%	87.5%	Montage-25	100%	87.5%	75%	62.5%	100%	62.5%
Inspiral-50	100%	50%	100%	50%	100%	100%	Inspiral-30	100%	25%	100%	50%	100%	100%
Sipht-60	100%	100%	100%	62.5%	100%	100%	Sipht-30	100%	100%	100%	62.5%	100%	100%
Epigenomics-46	100%	25%	100%	75%	100%	100%	Epigenomics-24	100%	0	100%	37.5%	100%	100%

equal to d_i, the algorithm meets the deadline constraint. This paper uses the success rate s^r, which is the percentage of deadlines met by an algorithm among all deadlines, to evaluate the capability of an algorithm to meet the deadline of a workflow. The calculation of s^r is as follows [17]:

$$s^r = \frac{\kappa}{\varphi} \times 100\% \tag{14}$$

where κ is the number of deadlines met by an algorithm, φ is the total number of deadlines. Table IV shows the success rate of each algorithm for each workflow.

The upper-left region of Table IV shows that HPSO, SPSO, ICPCP, and JIT have a 100% success rate for all extra-large workflows with about 1000 tasks, which means the eight deadlines of each extra-large workflow are met by them. LACO meets all deadlines of each extra-large workflow except Sipht-1000, for which the success rate of LACO is 87.5% because it fails to meet the tightest deadline d_1. For MEMS, its success rates for Inspiral-1000 and Montage-1000 are 12.5% and 75%, respectively, and the success rates for the three other extra-large workflows are 100%.

For the large workflows with about 100 tasks, as shown in the upper-right region of Table IV, the deadlines of all large workflows are satisfied by HPSO. ICPCP, LACO, and JIT can meet all deadlines of large workflows except CyberShake-100, for which the success rates are 87.5%, 87.5% and 62.5%, respectively. SPSO has a 100% success rate only for Montage-100 and Sipht-100. Except Epigenomics-100, MEMS does not have a 100% success rate for other large workflows.

The lower-left and lower-right regions of Table IV show that for medium and small workflows, the performance differences in meeting deadlines amongst HPSO, LACO, and the four other algorithms are more obvious. HPSO and LACO can meet all deadlines of medium and small workflows, whereas the other algorithms perform less well in meeting deadlines as the size of a workflow decreases. Amongst small workflows, the success rates of MEMS and JIT for CyberShake-30 and the success rates of SPSO for Epigenomics-24 are all 0. It is because the deadline of a workflow calculated by (13) is shortened with the decrease of a workflow's size, thus reflecting the capacity of different algorithms to meeting deadlines.

Overall, for all workflows with different sizes, HPSO has 100% success rates and is the best algorithm, followed by LACO, ICPCP, JIT, and SPSO, while MEMS is the worst. The main reasons of HPSO having good performance in meeting the deadlines are as follows: one is that idle time slots of leased instances are fully used, which means a task may be completed as early as possible; the second is multiple different task scheduling sequences are considered and used in the proposed method. LACO's performance is slightly inferior to HPSO because idle time slots of a leased instance are not considered. ICPCP uses the PCP as a whole scheduling object, which means an instance must satisfy *LFT* values of all tasks on the PCP, so it is not easy to meet the more urgent deadlines. When choosing the cheapest VM type for a task, JIT considers its effect on the children of the task. However, there may be no existing VM types that can meet the requirements of the task and its children, thus making JIT fail to the tightest deadline of some workflows. For SPSO and MEMS, they both randomly implement the mapping of tasks to VM instances during the iteration process, and do not consider the *LFT* of a task and the idle time slots of a leased instance.

2) Execution Cost Evaluation

The performance of an algorithm in minimizing the execution cost of a workflow is evaluated by the average execution cost of the workflow scheduled by the algorithm. Figs. 4–7 show the performance of comparison algorithms in minimizing the execution cost of extra-large, large, medium, and small workflows, respectively.

Each curve in the figures consists of eight points (cost values), reflecting the average execution cost of a workflow scheduled by an algorithm under different deadlines. In figures, the point circled by an ellipse is an invalid point, which indicates the algorithm fails to meet the corresponding deadline. In this case, comparing the average execution costs of valid points and invalid ones is meaningless.

Amongst extra-large workflows, the execution cost of CyberShake-1000 scheduled by HPSO is lower than those scheduled by the five other algorithms, which is shown in Fig. 4(a); for Inspiral-1000 shown in Fig. 4(b), HPSO and LACO have better performance than other algorithms, and LACO is

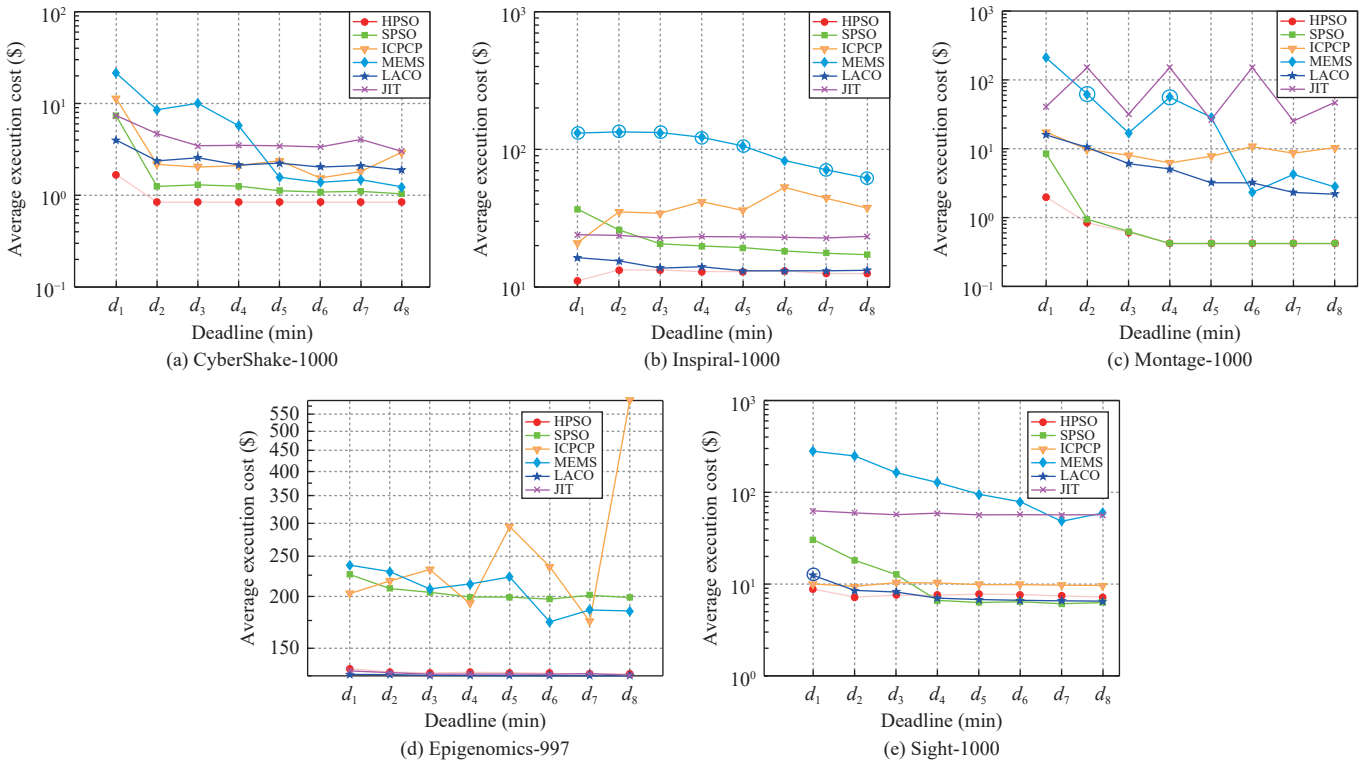

Fig. 4. Comparsion of six algorithms in terms of minimizing the execution cost for extra-large workflows.

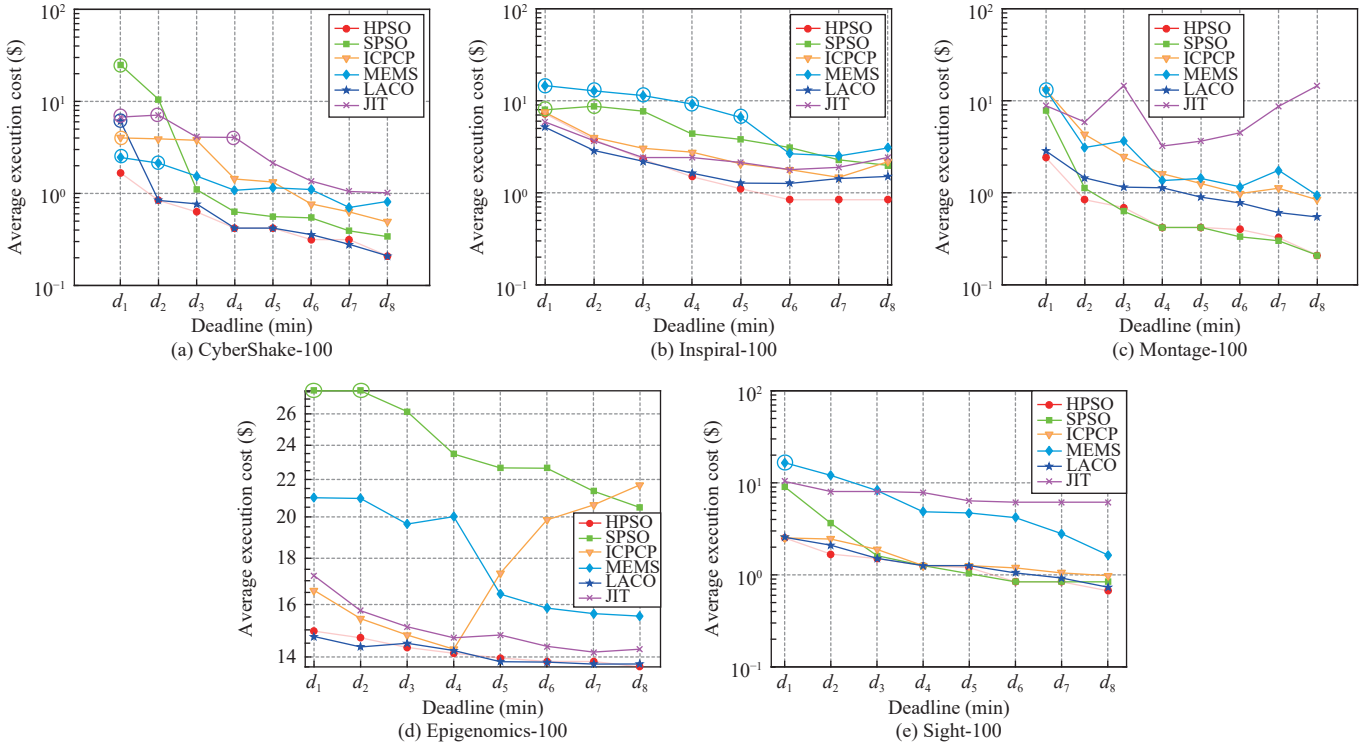

Fig. 5. Comparsion of six algorithms in terms of minimizing the execution cost for large workflows.

inferior to HPSO; for Montage-1000 shown in Fig.4(c), HPSO is slightly better than SPSO because the execution costs obtained by HPSO under the top two tightest deadlines are lower than those obtained by SPSO, and the performance of HPSO is significantly better than LACO and other algorithms; for Epigenomics-997 shown in Fig. 4(d), HPSO, LACO, and JIT perform almost the same and better than ICPCP, SPSO, and MEMS; for Sight-1000 and its top three tightest deadlines, as shown in Fig. 4(e), the execution costs obtained by HPSO are lower than those obtained by other algorithms. Overall, the performance of HPSO in terms of minimizing the execution cost of extra-large workflows is better than that of

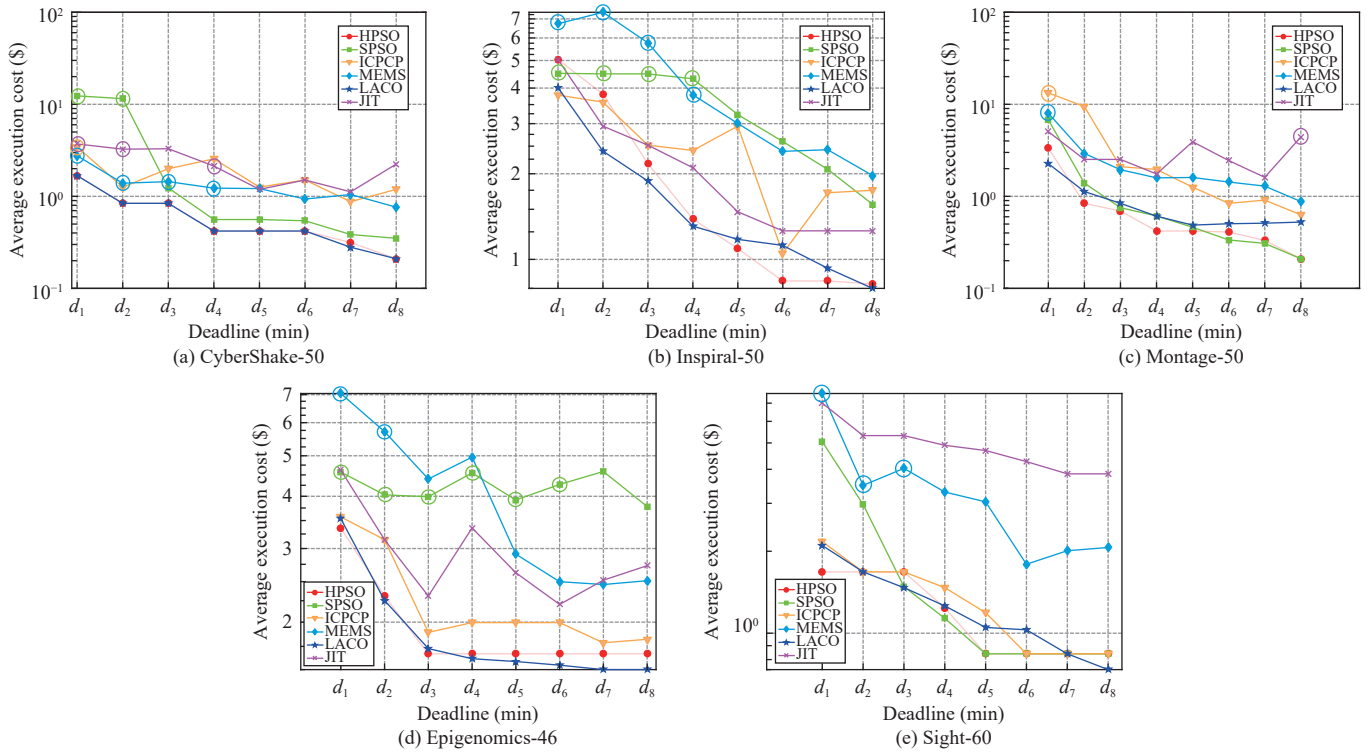

Fig. 6. Comparsion of six algorithms in terms of minimizing the execution cost for medium workflows.

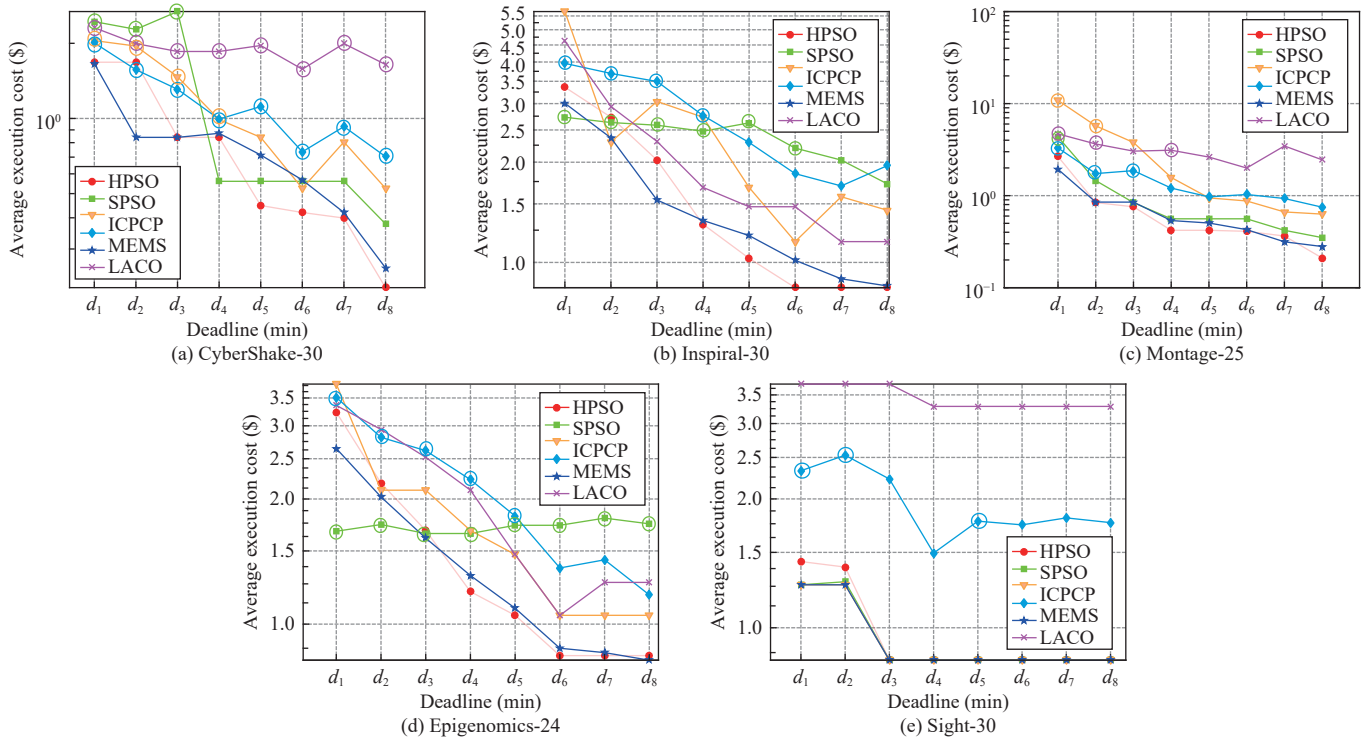

Fig. 7. Comparsion of six algorithms in terms of minimizing the execution cost for small workflows.

other algorithms.

Amongst the large workflows shown in Fig. 5, the execution cost of CyberShake-100 scheduled by LACO under the tightest deadline d_1 is significantly higher than that scheduled by HPSO, and LACO fails to meet the deadline d_1. For Inspiral-100, the execution costs obtained by HPSO under the deadlines from d_4 to d_8 are lower than those obtained by LACO. Thus for CyberShake-100 and Inspiral-100 shown in Figs. 5(a)–5(b), HPSO performs slightly better than LACO, and both of them are better than others; for Montage-100, HPSO has a similar performance to SPSO and is better than LACO and other algorithms. Fig. 5(d) shows that HPSO and

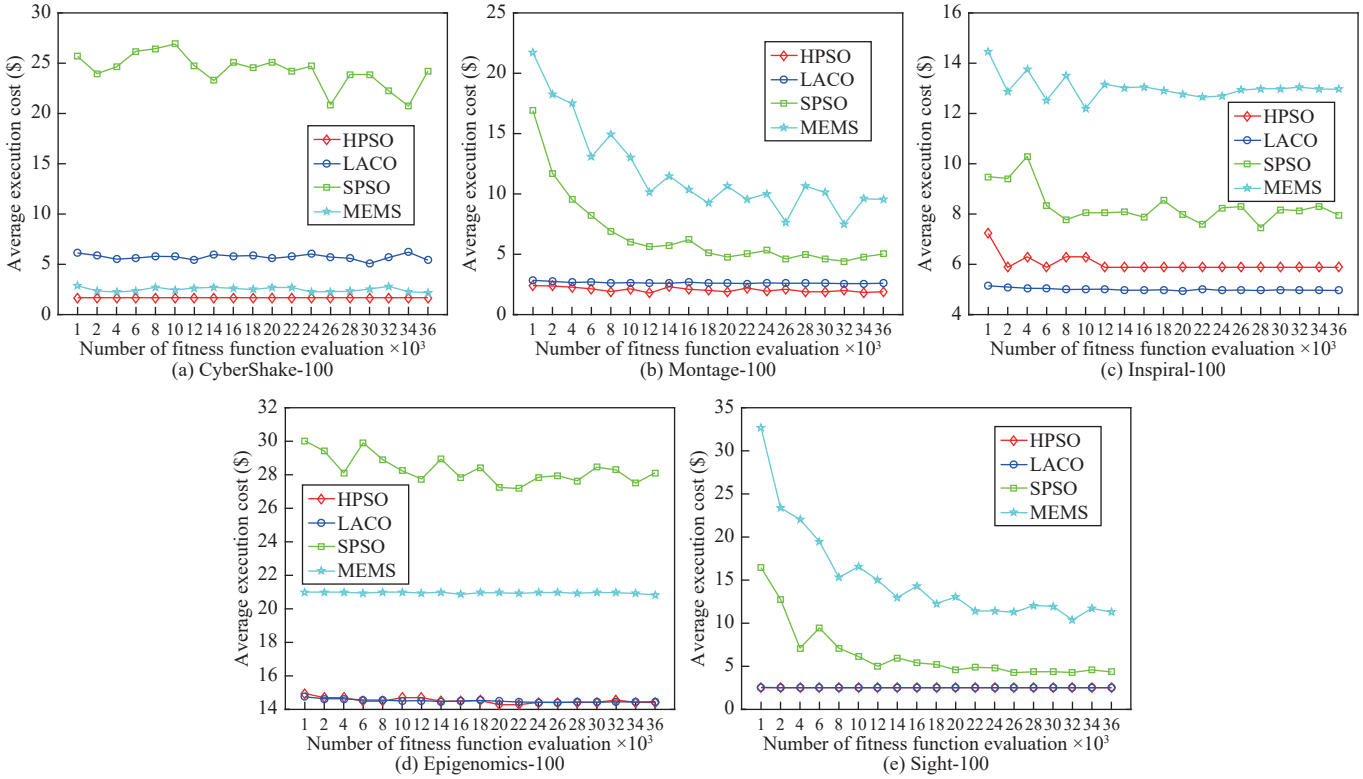

Fig. 8. Average evolutionary curves of HPSO, LACO, SPSO and MEMS on large workflows under the tightest deadline d_1.

LACO have similar performance for Epigenomics-100 and are better than SPSO and other algorithms. For Sipht-100, the execution cost obtained by HPSO, LACO, and SPSO is similar under the deadlines from d_3 to d_8. Overall, compared to LACO, HPSO can obtain lower execution costs for CyberShake-100, Inspiral-100, and Montage-100. Similarly, compared to SPSO, HPSO performs better for CyberShake-100, Inspiral-100, and Epigenomics-100. For all large workflows, HPSO performs better than ICPCP, JIT, and EMES.

Amongst the medium and small workflows shown in Figs. 6–7, the execution costs obtained by HPSO and LACO for CyberShake-50, Epigenomcis-46/24, and Sipht-60/30 are similar; for Inspiral-50/30 shown in Fig. 6(b) and Fig. 7(b), the execution cost obtained by HPSO under deadlines d_1–d_4 is higher than that obtained by LACO, while the cost obtained by HPSO under d_5–d_8 is lower than that obtained by LACO. For Montage-50/25 shown in Fig. 6(c) and Fig. 7(c), HPSO performs better than LACO. Although Figs. 6–7 show that for medium and small workflows LACO performs similar with HPSO, HPSO can obtain lower execution costs than LACO for extra-large and large workflows (see Figs. 4–5). That is, as the size of the workflow increases, the performance of HPSO is better than that of LACO. Furthermore, HPSO can meet deadlines better than LACO for extra-large and large workflows (see Table IV).

In general, HPSO can achieve a lower execution cost for most workflows amongst the six algorithms, followed by LACO, SPSO and ICPCP, and JIT and MEMS. The rule based heuristic scheduling algorithms ICPCP and JIT can find quickly a feasible solution for each workflow, but the solution quality is lower than HPSO, LACO, and SPSO for most workflows. MEMS performs worse because the execution costs obtained in its different runs are significantly different, resulting in high average cost. HPSO has the best performance because HPSO combines PSO and idle time-aware rules, having the advantage of both meta-heuristics (PSO) and heuristic rules.

3) Convergence of HPSO, LACO, SPSO and MEMS

To observe the convergence of HPSO, LACO, SPSO, and MEMS, along with the increase of the number of fitness function evaluations, Fig. 8 shows the average evolutionary curves of the four algorithms on large workflows under the tightest deadline d_1.

Fig. 8 shows that HPSO and LACO converge for almost all large workflows when the number of fitness function evaluations K reaches 1000. For CyberShake-100 and Epigenomics-100, MEMS converges when K reaches 1000, while for other large workflows MEMS converges when K is about 10 000. Similarly, SPSO converges for all large workflows when K is about 10 000.

Evolutionary curves of LACO show that LACO cannot find solutions with lower cost values as the number of fitness function evaluations increases. This means that the performance of LACO cannot be better than that of HPSO even if a larger number of evaluations are given.

4) Impact of Task Scheduling Sequence on Execution Cost

In order to investigate the impact of multiple task scheduling sequences and a single task scheduling sequence on the execution cost of workflow, some modifications are made to HPSO. The modified HPSO only uses a single task scheduling sequence, which is obtained by topologically

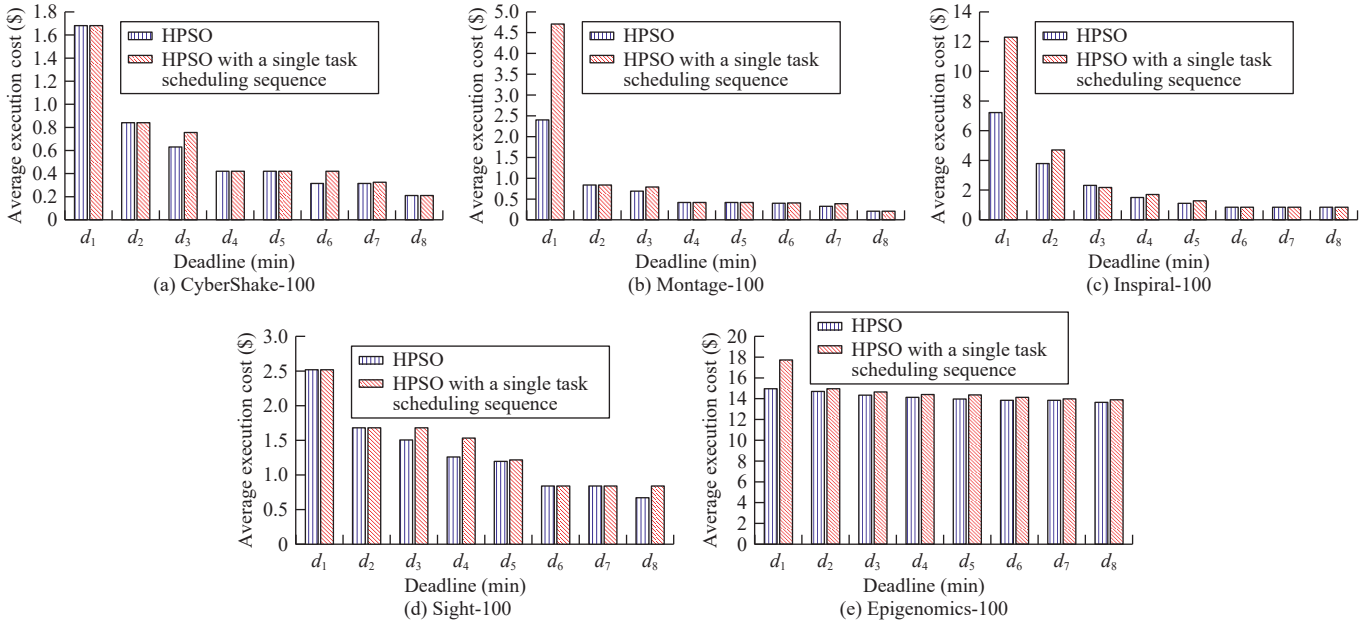

Fig. 9. Comparison of HPSO and HPSO with a single task scheduling sequence in terms of minimizing the execution cost.

sorting all tasks of a workflow. Therefore, each particle in HPSO with a single task scheduling sequence only needs to represent the mapping of tasks to VM types, not including the priority of each task. Fig. 9 gives the performance comparison between HPSO and HPSO with a single task scheduling sequence, in terms of minimizing the execution cost of each large workflow under each deadline. In the figure, "HPSO" refers to HPSO with multiple task scheduling sequences.

Fig. 9 shows that under each deadline of each large workflow, the average execution cost obtained by HPSO is lower than or equal that obtained by HPSO with a single task scheduling sequence. For example, in Fig. 9(b), 9(c), and 9(e), the average execution cost of Montage-100, Insiral-100, and Epigenomics-100 under the tightest deadline d_1 scheduled by HPSO is significantly lower than that scheduled by HPSO with a single task scheduling sequence. Overall, HPSO with multiple different tasks scheduling sequences can achieve lower execution cost than HPSO with a single task scheduling sequence.

5) Comparison of Runtime

Compared with heuristic rules, meta-heuristic based scheduling algorithms usually need a longer computational time. This section compares the runtime of HPSO, SPSO, MEMS, and LACO. Fig. 10 gives the average runtime of each algorithm for different workflow sizes (30, 50, 100, and 1000). The runtime is calculated by first summing up the running time of an algorithm for each workflow instance and then divided by the total number of workflow instances. All four algorithms are coded in MATLAB R2016a and implemented on a PC with Core i7 2.50 GHz and Windows 7 operation system.

Fig. 10 shows that the average runtime of HPSO and LACO is less than that of SPSO and MEMS for workflows with 30 and 50 tasks, while the average runtime of HPSO for extra-large workflows is longer than that of LACO, SPSO, and MEMS. For workflows with 100 tasks, the average runtime of

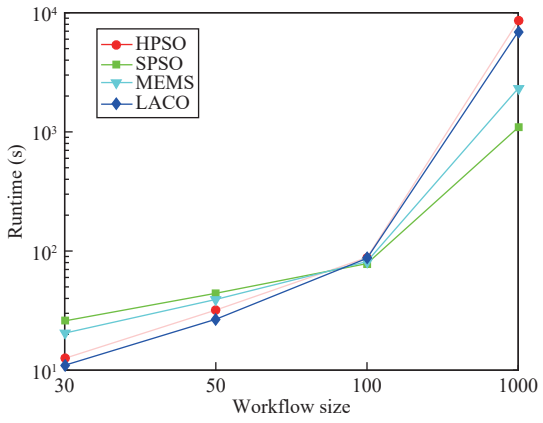

Fig. 10. The average runtime of HPSO, SPSO, MEMS, and LACO.

the four algorithms is very close.

VI. Conclusions and Future Work

This paper proposes a hybrid particle swarm optimization (HPSO) for cloud workflow scheduling problem that considers three aspects, namely selecting the most appropriate VM type for each task, determining the best scheduling sequence for all tasks, and making full use of idle time slots. The first two aspects are combinational optimization problems, such that a PSO is used to solve them. In PSO, a particle represents the VM type required by each task and the scheduling sequence of tasks. Idle time slot-aware scheduling rules are proposed to decode a particle to a scheduling solution. During the decoding procedure, the rules assign each task in the order of the scheduling sequence to a leased VM instance or a new one, taking full use of idle time slots of leased VMs. As the randomness of PSO may cause priorities of some tasks to be invalid, making the task scheduling sequence violate the tasks' precedence constraint, a simple repairing method is suggested to repair invalid priorities of

tasks.

Experimental results based on synthetic workflows show that HPSO achieves 100% success rates in meeting workflows' deadlines and outperforms the five other comparison algorithms. Moreover, compared with other algorithms, HPSO has better performance in minimizing the execution cost for most workflow applications.

In the future, we plan to implement a prototype system to further test HPSO. In addition, it is hard to predict task execution times exactly in advance. In [48], [49], a cloud system is modeled as a discrete-time-state space to deal with non-steady states of cloud system. Based on the idea of the discrete-time-state cloud system, we plan to combine HPSO with prediction technologies (such as analytic probabilistic models or deep learning) to study dynamic cloud workflow scheduling algorithms that considers the uncertainty of task execution times.

REFERENCES

[1] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities," *Futur. Gener. Comp. Syst.*, vol. 25, no. 5, pp. 528–540, May 2009.

[2] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*, document SP 800–145, NIST, Gaithersburg, MD, USA, 2001.

[3] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: A survey," *J. Supercomput.*, vol. 71, no. 9, pp. 3373–3418, 2015.

[4] Amazon elastic compute cloud (Amazon EC2) [Online]. Available: http://aws.amazon.com/ec2/, Accessed on: Mar. 2020.

[5] J. Sahni and D. 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, Jan.-Mar. 2018.

[6] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Inf. Sci.*, vol. 270, pp. 255–287, Jun. 2014.

[7] Ullman and D. Jeffrey, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, 1975.

[8] Z. H. Zhan, X. F. Liu, Y. J. Gong, J. Zhang, S.H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–33, Jul. 2015.

[9] 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, May 2016.

[10] 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, "Multi-objective cloud workflow scheduling: a multiple populations ant colony system approach," *IEEE Trans. Cloud Comput.*, vol. 49, no. 8, pp. 2912–2926, Aug. 2019.

[11] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, " A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Proc. IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Perth, WA, Australia, 2010, pp. 400–407.

[12] H. Y. Hu, Z. J. Li, H. Hu, J. Chen, J. D. Ge, C. Y. Li, and V. Chang, "Multi-objective scheduling for scientific workflow in multicloud environment," *J. Netw. Comput. Appl.*, vol. 114, pp. 108–122, Jul. 2018.

[13] H. R. Faragardi, S. Dehnavi, T. Nolte, M. Kargahi, and T. Fahringer, "An energy-aware resource provisioning scheme for real-time applications in a cloud data center," *Softw., Practice Experience*, vol. 48, no. 10, pp. 1734–1757, 2018.

[14] X. Xu, W. Dou, X. Zhang, and J. Chen, " EnReal: An energy-aware resource allocation method for scientific workflow executions in cloud environment," *IEEE Trans. Cloud Comput.* , vol. 4, no. 2, pp. 166–179, Apr.–Jun. 2016.

[15] M. S. Kumar, I. Gupta, and P. K. Jana, " Resource-aware energy efficient workflow scheduling in Cloud Infrastructure," in *Proc. IEEE Int. Conf. Adv. Comput., Commun. Inf.*, Bangalore, India, 2018, pp. 293–299.

[16] H. R. Faragardi, M. R. Saleh Sedghpour, S. Fazliahmadi, T. Fahringer, and N. Rasouli, " GRP-HEFT: A budget-bonstrained resource provisioning scheme for workflow scheduling in IaaS clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1239–1254, Jun. 2020.

[17] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget and deadline aware e-science workflow scheduling in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 29–44, Jan. 2019.

[18] M. A. Rodriguez and R. Buyya, "Budget-driven scheduling of scientific workflows in IaaS clouds with fine-grained billing periods," *ACM Trans. Auton. Adapt. Syst.*, vol. 12, no. 2, pp. 1–22, May 2017.

[19] H. Chen, X. Zhu, D. Qiu, L. Liu and Z. Du, "Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds," *IEEE Trans. Parallel Distrib. Syst.* , vol. 28, no. 9, pp. 2674–2688, Sept. 2017.

[20] Z. J. Li, J. D. Ge, H. J. Yang, L. G. Huang, H. Y. Hu, H. Hu, and B. Luo, "A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds," *Futur. Gener. Comp. Syst.*, vol. 65, pp. 140–152, Dec. 2016.

[21] 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, Apr.–Jun. 2014.

[22] L. Liu, M. Zhang, R. Buyya, and Q. Fan, "Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing," *Concurr. Comput.-Pract. Exp.*, vol. 20, no. 5, pp. 1–12, Mar. 2017.

[23] Z. G. Chen, K. J. Du, Z. H. Zhan, and J. Zhang, "Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm," in *Proc. IEEE Congr. Evol. Comput.*, Sendai, Japan, 2015, pp. 708–714.

[24] Z. G. Chen, Z. H. Zhan, H. H. Li, K. J. Du, J. H. Zhong, Y. W. Foo, Y. Li, and J. Zhang, " Deadline constrained cloud computing resources scheduling through an ant colony system approach," in *Proc. IEEE Int. Conf. Cloud Comput. Res. Innov.*, Singapore, 2015, pp. 112–119.

[25] Y. H. Jia, W. N. Chen, H. Q. Yuan, T. L. Gu, H. X. 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.*, to be published, DOI: 10.1109/TSMC.2018.2881018

[26] P. Kaur and S. Mehta, "Resource provisioning and workflow scheduling in clouds using augmented Shuffled Frog Leaping Algorithm," *J. Parallel Distrib. Comput.*, vol. 101, pp. 41–50, 2017.

[27] Z. Tong, H. J. Chen, X. M. Deng, K. L. Li, and K. Q. Li, "A novel task scheduling scheme in a cloud computing environment using hybrid biogeography-based optimization," *Soft Comput.* , vol. 23, pp. 11035–11054, 2019.

[28] X. Zuo, G. Zhang and W. Tan, " Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 564–573, Apr. 2014.

[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, Dec. 2017.

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

[31] 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.

[32] 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, Jan. 2013.

[33] V. Arabnejad, K. Bubendorfer, and B. Ng, " Scheduling deadline constrained scientific workflows on dynamically provisioned cloud resources," *Futur. Gener. Comp. Syst.*, vol. 75, pp. 348–364, Oct. 2017.

[34] R. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 7, pp. 1787–1796, Jul. 2014.

[35] X. Li and Z. Cai, " Elastic resource provisioning for cloud workflow applications," *IEEE Trans. Autom. Sci. Eng.* , vol. 14, no. 2, pp. 1195–1210, April. 2017.

[36] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-aware online scheduling for real-time workflows in cloud service environment," *IEEE Trans. Serv. Comput*. to be published, DOI: 10.1109/TSC.2018.2866421.

[37] X. M. Zhou, G. X. Zhang, J. Sun, J. L. Zhou, T. Q. Wei, and S. Y. Hu, "Multi-objective workflow scheduling in Amazon EC2," *Cluster Comput.*, vol. 17, pp. 169–189, 2014.

[38] X. M. Zhou, G. X. Zhang, J. Sun, J. L. Zhou, T. Q. Wei, and S. Y. 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.

[39] Y. K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[40] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, *et al*., " A performance analysis of EC2 cloud computing services for scientific computing," in *Proc. Lect. Notes Inst. Comput. Sci. Soc. Informatics Telecommun. Eng.*, Berlin, Heidelberg, Germany, 2009, pp. 115–131.

[41] Y. Shi and R. C. Eberhart, " Empirical study of particle swarm optimization", in *Proc. IEEE Congr. Evol. Comput.*, Washington DC, USA, 1999, pp.1945–1950.

[42] L. Tang and X. Wang, " An Improved Particle Swarm Optimization Algorithm for the Hybrid Flowshop Scheduling to Minimize Total Weighted Completion Time in Process Industry," *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 6, pp. 1303–1314, Nov. 2010.

[43] J. Bi, H. Yuan, Y. Fan, W. Tan, and J. Zhang, "Dynamic fine-grained resource provisioning for heterogeneous applications in virtualized cloud data center," in *Proc. IEEE Int. Conf. on Cloud Computing*, New York, NY, USA, 2015, pp. 429–436.

[44] H. Yuan, J. Bi, B. H. Li, and W. Tan, "Cost-aware request routing in multi-geography cloud data centres using software-defined networking," *Enterp. Inf. Syst.*, vol. 11, no. 3, pp. 359–388, Mar. 2017.

[45] K. Deb, A. Pratap, S. Agarwal, and T. A. M. T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[46] 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, Mar. 2013.

[47] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, " CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw.*: *Practice Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[48] H. T. Yuan, J. Bi, M. C. Zhou, Q. Liu, and A. C. Ammari, "Biobjective task scheduling for distributed green data centers," *IEEE Trans. Autom. Sci. Eng*., to be published, DOI: 10.1109/TASE.2019.2958979.

[49] J. Bi, H. T. Yuan, W. Tan, M. C. Zhou, Y. S. Fan, J. Zhang, and J. Q. Li, " Application-Aware Dynamic Fine-Grained Resource Provisioning in a Virtualized Cloud Data Center," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 2, pp. 1172–1184, Apr. 2017.

Yun Wang received the B.Sc. degree in computer science from Sichuan University, China, in 2001, and the M.Sc. degree in computer science from Nanchang University, China, in 2008. She is currently a Ph.D. candidate in computer science and technology at the School of Computing Science, Beijing University of Posts and Telecommunications, China. She is also currently an Associate Professor with the School of Information Engineering, Nanchang Institute of Technology, China. Her main research interests include workflow scheduling, cloud computing, and evolutionary computation.

Xingquan Zuo (SM' 14) received the Ph.D. degree in control theory and control engineering from Harbin Institute of Technology, Harbin, China, in 2004. He is currently a Professor in the School of Computer Science, Beijing University of Posts and Telecommunications, China. From 2004 to 2006, he was a Postdoctoral Research Fellow in Automation Department, Tsinghua University, China. From 2012 to 2013, he was a Visiting Scholar in Industrial and System Engineering Department, Auburn University, AL, USA. His research interests include intelligent optimization and scheduling, data mining, artificial intelligence, and intelligent transportation systems. He has published over 100 research papers in journals and conferences, two books and several book chapters. He is a Senior Member of IEEE, Senior Member of China Computer Federation, and Senior Member of Chinese Association for Artificial Intelligence. He is Committee Member of Intelligent Simulation Optimization and Scheduling Society and Transportation Model and Simulation Society of Chinese Association for System Simulation, and Committee Member of Intelligent Service Society of Chinese Association for Artificial Intelligence. He served in Program Committee or Program Chair of more than 10 international conferences.