Cost-Driven Scheduling for Deadline-Based Workflow Across Multiple Clouds

Wenzhong Guo, Bing Lin, Guolong Chen, Yuzhong Chen, and Feng Liang

***Abstract*—With the development of cloud computing, the coexistence of multiple cloud service providers appears in the current cloud market. Due to heterogeneous instance types, different bandwidths and various price models among multiple clouds, it is a challenging issue to schedule a deadline-constrained scientific workflow across multiple clouds. Existing research for workflow scheduling are mostly in the traditional distributed computing environment (such as grid), and only a few primal contributions are made in the cloud environment. This paper proposes a scheduling strategy for a deadline-constrained scientific workflow across multiple clouds. In order to minimize the execution cost of the workflow while meeting its deadline, our strategy utilizes the discrete particle swarm optimization technique, and adopts randomly two-point crossover operator and randomly single point mutation operator of the genetic algorithm. Besides, the strategy optimizes the performance for both computation cost and data transfer cost across multiple clouds. Our strategy is evaluated through well-known workflows, and experimental results show that it performs better than other state-of-the-art strategies.**

***Index Terms*—Cloud computing, workflow scheduling, deadline constraints, cost optimization.**

I. Introduction

SCIENTIFIC applications are usually data-intensive and computation-intensive, and they are composed of hundreds of interrelated tasks [1]. Workflow model [2], [3] has been an effective way to represent such complicated scientific applications. Montage [4], for example, used several workflows to generate a set of custom mosaics of sky created by NASA. The workflows employed in scientific fields usually consist of several tasks connected with each other through data/control dependencies [3]. It is challenging to schedule tasks on their appropriate resources with data/control dependencies in a workflow.

Workflow scheduling has been broadly studied for many years in traditional environments [5]–[8], such as clusters and Grid. Most of the previous works aggressively pursued the minimum workflow makespan (i.e., total execution time) and the maximum resource utilization. With the rapid development of cloud computing, many cloud service providers have appeared in recent years, such as Amazon EC2 [9], Rackspace [10] and GoGrid [11]. Cloud resources are charged on demand, and resource acquisition in the cloud environment considers both performance metrics and commercial costs [12]. There are a few works [4], [5] addressing workflow scheduling while considering commercial cost in the cloud environment. In addition, most of them ignored fundamental principles of cloud computing, such as the elasticity [13] and the heterogeneity [14] of cloud resources. It is difficult to schedule a workflow in the cloud environment while considering basic cloud principles.

The fluctuation of virtual machine (VM) performance increases the challenge of scheduling workflow on cloud platforms [5]. Schad *et al.* [15] reported that the overall variability of CPU performance was 24% on Amazon's EC2. It will cause the workflow to miss its deadline and has a great impact on scheduling decision. Many previous works made a task mapping decision relying on the estimated task runtime [6], [16], [17]. However, the actual task runtime is always greater than the estimated runtime [15]. This delay will influence successors of each task and will not disappear until the whole workflow finishes executing.

The deadline factor has a significant impact on the scheduling strategy for scientific applications [18]. Different cloud service providers exhibit a set of diversities regarding price mechanisms, instance types and charge time interval, etc. It usually has slow inter-bandwidth between two cloud providers and fast intra-bandwidth within a cloud provider [5]. Moreover, it will take certain fees to transfer data from a cloud provider to another one. When scheduling a workflow across multiple cloud providers, it should consider not only diversities of cloud services, but also different bandwidths and data transfer cost.

Rodriguez and Buyya [5] proposed discrete particle swarm optimization (PSO)-based strategy for scheduling a workflow in a single cloud. It utilized a global optimization technique to reduce the overall execution cost of a workflow while

Manuscript received December 14, 2017; revised June 15, 2018 and September 11, 2018; accepted September 20, 2018. Date of publication September 28, 2018; date of current version December 10, 2018. This work is partly supported by the National Natural Science Foundation of China under Grants No. 61672159 and No. U1705262. The associate editor coordinating the review of this paper and approving it for publication was Y. Diao. *(Corresponding author: Bing Lin.)*

W. Guo, G. Chen, and Y. Chen are with the College of Mathematics and Computer Sciences, Fuzhou University, Fuzhou 350108, China, also with the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, Fuzhou 350108, China, and also with the Fujian Collaborative Innovation Center for Big Data Applications in Governments, Fuzhou University, Fuzhou 350003, China (e-mail: fzugwz@163.com; cgl@fzu.edu.cn; yzchen@fzu.edu.cn).

B. Lin is with the College of Physics and Energy, Fujian Normal University, Fuzhou 350117, China, and also with the Fujian Provincial Collaborative Innovation Center for Optoelectronic Semiconductors and Efficient Devices, Fujian Normal University, Xiamen 361005, China (e-mail: wheellx@163.com).

F. Liang is with Department of Computer Science, University of Hong Kong, Hong Kong (e-mail: loengf@connect.hku.hk).

Digital Object Identifier 10.1109/TNSM.2018.2872066

satisfying its deadline constraint. Inspired by this work, our strategy is also based on the PSO algorithm, which tends to be effective in addressing NP-hard problems [19].

In this study, we develop an adaptive discrete PSO strategy with genetic algorithm operators (ADPSOGA) for scheduling a deadline-constrained scientific workflow across multiple clouds. The main contributions of this study are summarized as follows:

- Our strategy takes into account the more essential characteristics of multiple clouds, such as the data transfer cost among different cloud providers, the boot time and shutdown time of VMs, and the inter-bandwidth between different cloud providers.
- To avoid the premature convergence problem of traditional PSO, we introduce the randomly two-point crossover operator and randomly single mutation operator of the genetic algorithm (GA), which can increase the diversity of evolving populations.
- A new scheduling strategy is proposed to minimize the execution cost of workflow while meeting its deadline constrains across multiple clouds. This strategy optimizes both data transfer cost and computation cost.

The rest of this study is organized as follows: some related work is presented in Section II. Section III describes the scheduling model in detail, and then Section IV represents the proposed ADPSOGA algorithm. In Section V, we compare our strategy with other state-of-the-art strategies. Finally, Section VI makes some concluding remarks.

II. Related Work

Workflow scheduling in traditional distributed systems, such as grid, has been investigated in depth in recent years. It is a NP-hard problem [20], and there is no polynomial way to obtain an optimal result unless P = NP. Hence, meta-heuristic and heuristic strategies were employed to acquire an approximately optimal solution. The optimal object for workflow scheduling in the grid was usually to minimize the makespan of workflow. Cao *et al.* [6] proposed an innovative workflow scheduling strategy based on list scheduling and group scheduling strategies. This strategy could decrease the execution time of workflow and improve the resource utilization in the grid environment. Chen and Zhang [7] designed a workflow scheduling strategy based on Ant Colony Optimization (ACO). They pursued to optimize the workflow execution time and satisfy QoS requirements in the grids. In addition, many works took into account the workflow budget. Khajemohammadi *et al.* [8] presented a multi-objective genetic algorithm to optimize both workflow cost and makespan. They generated a feasible solution through developing a multi-level workflow model. Yu and Buyya [16] also proposed a new GA-based technique for workflow scheduling in utility grids, but these proposed strategies aim at minimizing the workflow makespan while satisfying a defined-budget constraint. Another cost-driven scheduling strategy for workflow in the grid environment was proposed by Abrishami *et al.* [17]. They aimed to minimize the execution cost of workflow based on the strategy of scheduling all the tasks in a partial critical path (PCP) on the specific VM.

The aforementioned strategies in the grid environment afford a precious insight into the opportunities and challenges for the workflow scheduling. However, there is a big gap between grid and cloud in terms of resource provision mode. Zhu *et al.* [21] presented an iterated heuristic framework to schedule a workflow to elastic hybrid cloud resources, whose optimization objectives are explored: number, usage time and utilization of rented VMs. Malawski *et al.* [22] represented both dynamic algorithm and static algorithm for scheduling workflow ensembles in Infrastructure-as-a-Service (IaaS) clouds. They attempted to improve the completion rate of workflow ensembles while meeting both budget and deadline constraints. They realized that the leased VM needed some time to be ready for executing tasks. There was only one type of VM in their work, and this assumption did not conform to the reality of current cloud environment.

Mao and Humphrey [18] also focused on the workflow ensembles scheduling in the clouds. There are many VMs with different price/performance ratios for executing the tasks. A dynamic strategy was represented to optimize the total execution cost of workflow ensemble according to the pay-as-you-go model. Although this was an efficient strategy for minimizing the total execution cost of workflow ensemble, the designed solution was not an optimal strategy for a single workflow.

In recent years, many works [5], [13], [23] have adopted PSO to address the workflow scheduling issue, and obtained significant results. Wu *et al.* [23] represented a PSO-based strategy for scheduling a single workflow in the clouds. They aimed to reduce either makespan or cost while satisfying either budget or deadline constraint. They acknowledged that there were various VM types in the cloud environment. They assumed that a number of initialized VMs were available in advance and ignored the elasticity of resource provision. Pandey *et al.* [13] also proposed a PSO-based strategy to reduce the cost of scheduling a workflow in the clouds. However, they also ignored the elasticity of resource provision. More in line with our work was the proposed strategy for scheduling a single workflow in [5]. This strategy took into account the fundamental cloud characteristics regarding the heterogeneity of VMs, the leased delay of VMs and price model. It utilized a global optimization technique to reduce the overall execution cost of workflow while satisfying the deadline constraint. However, this work only considered the cost-driven scheduling for deadline-constrained workflow in a single cloud. It ignored the bandwidth difference and data transfer cost during workflow scheduling.

The definition of multi-cloud computing was first represented by Keahey *et al.* [24]. Fard *et al.* [25] designed a workflow scheduling strategy in commercial multi-cloud environment. This strategy optimized both makespan and cost of workflow. They tailored their strategy for scheduling a dynamic workflow suitably and obtained a prefect result. Although they took into account the elasticity of resource provision, the heterogeneity of cloud resource was ignored in their scheduling model. Zhang [26] proposed a resource scheduling

TABLE I
THE ANALYSIS OF RELATED WORKS ON WORKFLOW SCHEDULING

Reference	Infrastructure			Application type			Fluctuation		Object		Constraint		Strategy		
	Grid	Single-cloud	Multi-clouds	Jobs	Workflow	Workflow ensemble	Static	Dynamic	Cost	Time	Deadline	Budget	Mathematical programming	Heuristics	Meta-heuristics
Rodriguez et al. '14 [5]	-	+	-	-	+	-	-	+	+	-	+	-	-	-	+
Cao et al. '10 [6]	+	-	-	-	+	-	+	-	-	+	-	-	+	-	-
Chen et al. '09 [7]	+	-	-	-	+	-	+	-	+	+	-	-	-	+	-
Khajemohammadi et al. '13 [8]	+	-	-	-	+	-	+	-	+	+	-	-	+	-	-
Pandey et al. '10 [13]	-	+	-	-	+	-	+	-	+	+	-	-	-	-	+
Yu et al. '06 [16]	+	-	-	-	+	-	+	-	-	+	-	+	-	+	-
Abrishami et al. '12 [17]	+	-	-	-	+	-	+	-	+	-	+	-	-	+	-
Mao et al. '11 [18]	-	+	-	-	-	+	-	+	+	-	+	-	-	+	-
Zhu et al. '13 [21]	-	-	+	-	+	-	+	-	-	+	-	-	-	+	-
Malawski et al. '12 [22]	-	+	-	-	-	+	+	-	-	-	+	+	+	+	-
Wu et al. '10 [23]	-	+	-	-	+	-	+	-	+	+	+	+	-	-	+
Fard et al. '13 [25]	-	-	+	-	+	-	-	+	+	+	-	-	-	+	-
Li et al. '12 [26]	-	-	+	-	+	-	+	-	-	+	-	-	-	+	-
Lin et al. '16 [27]	-	-	+	-	+	-	+	-	+	-	+	-	-	+	-
Our work	-	-	+	-	+	-	-	+	+	-	+	-	-	-	+

strategy for workflow in wireless small clouds. This work aimed to finish executing all tasks in a workflow at the earliest time, and reduce the data transfer time between tasks executed on different resources.

We have proposed a workflow scheduling strategy [27], which also aimed to reduce the execution cost of workflow while satisfying its deadline constraint across multiple clouds. This strategy adopted the PCP strategy and assigned sub-deadlines to the PCPs instead of independent tasks. This operation could reduce the data transfer time. In addition, the strategy tried to find a best-fit instance to execute the entire PCP before its latest finish time. This operation could reduce the execution cost of workflow in commercial cloud environment. This work adopted heuristics algorithm but not meta-heuristics algorithm in this paper, and it did not consider the performance variation in the fluctuant environment.

The analysis of related works and our work has been summarized in Table I. The strategies are divided into 6 categories. The infrastructure types contain a grid, a cloud, or multi-clouds, where the grid includes heterogeneous computing systems and utility grids. The application types can be jobs, single workflow, or workflow ensemble. The fluctuation factors are divided into static and dynamic factors, where static factor means that it is constant during execution. We only consider two related optimization objectives: cost and time. The most commonly addressed constraints are deadline and budget in scheduling strategies. Finally, we classify the scheduling strategies into mathematical programming, heuristics, or meta-heuristics.

III. WORKFLOW SCHEDULING FRAMEWORK

Fig. 1 shows the workflow scheduling framework across multiple clouds. There are three main components in our scheduling framework, i.e., a deadline-constrained workflow, multi-cloud environments, and a cost-driven scheduler.

A workflow for the scientific application can be modeled as a directed acyclic graph (DAG) $G = (V, E)$, where

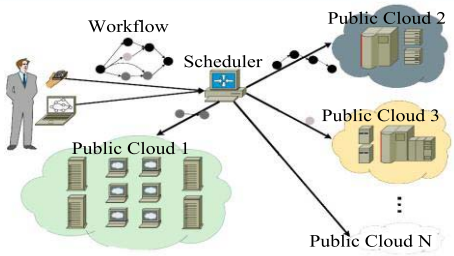

Fig. 1. Workflow scheduling framework across multiple clouds.

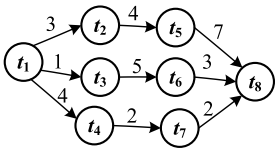

Fig. 2. Sample workflow.

$V = \{t_1, t_2, \ldots, t_n\}$ is defined as a finite set of n tasks, and $E = \{e_{ij} = (t_i, t_j) | t_i, t_j \in V\}$ is the set of directed arcs. Each directed arc e_{ij} indicates that there is a data dependency between task t_i and task t_j, and task t_j cannot be executed until task t_i is finished. For an arc $e_{ij} = (t_i, t_j)$, the task t_i is the direct parent of task t_j, and the task t_j is the direct child of task t_i. Fig. 2 displays a sample workflow with data dependency arcs, the number on which represents the amount of transferred data between each pair of tasks. In addition, a deadline constraint, $D(G)$, is associated with its workflow. A scheduling strategy is called the feasible solution when a workflow is finished before its deadline according this strategy.

There are various cloud service providers $P = \{p_1, p_2, \ldots, p_r\}$, and each provider p_i offers different types of VMs. vm_{ijk} is defined as the k^{th} launched VM with j^{th} type offered by provider p_i across multiple clouds. Each VM type has its particular processing capacity and memory storage.

According to the VM memory consumptions obtained from Amazon EC2 [28], we assume that each VM has enough memory to process the workflow. This assumption means that we only consider the processing capacity (i.e., CPUs) of VMs. A task t_i has an estimated execution time T_{exe} (t_i, vm$_{ijk}$) on vm_{ijk}, and each VM type from various providers has different price/performance ratios for a given task [29]. We use serial processing model, which means that a task is executed only on one VM and a VM can only execute one task at the same time.

A new launched VM vm_{ijk} needs some boot time T_{boot} (vm_{ijk}) to be available for any tasks. This initialization time has a great impact on workflow optimization scheduling [30]. A running VM does not shut down immediately until it completes all the tasks on it and transfer generated data to other VMs [5]. In addition, the leased VM offered by a provider p_p is charged based on a specific time unit λ_p, and any partial execution time should be considered as a full time period. For instance, $\lambda_p = 60$ minutes, if a VM is employed for 121 minutes, the user should pay for 3 hours (i.e., 180 minutes). Also, each VM vm_{pkr} with k^{th} type offered by provider p_p has a corresponding cost c_{pk} per time unit.

Due to the infrastructure from the same provider is mostly kept in a local region and different providers' VMs are reserved in various areas [31], we assume that the inter-bandwidth among different providers is slower than the intra-bandwidth in a single provider. The intra-data transfer time, T_{intra} (e_{ij}, p_p), between task t_i and its child t_j in provider p_p is calculated as formula (1), and formula (2) describes the inter-data transfer time, T_{inter} (e_{ij}, p_p, p_q), between provider p_p and provider p_q, respectively. *Data*(e_{ij}) is the amount of transferred data between task t_i and its child t_j. B_{intra} (p_p) is defined as the intra-bandwidth in provider p_p and B_{inter} (p_p, p_q) represents the inter-bandwidth between provider p_p and provider p_q, respectively. For simplicity, the intra-bandwidth among VMs is roughly the same, and the inter-bandwidth has no difference with each other [25]. The bandwidth in a single VM is assumed to be infinite [5], therefore the T_{intra} (e_{ij}, p_p) will be 0 if task t_i and its child t_j are both scheduled on the same VM.

$$T_{intra}(e_{ij}, p_p) = \frac{Data(e_{ij})}{B_{intra}(p_p)}, \tag{1}$$

$$T_{inter}(e_{ij}, p_p, p_q) = \frac{Data(e_{ij})}{B_{inter}(p_p, p_q)}. \tag{2}$$

Data transfer cost between two different providers will affect the final scheduling decisions [27], [32]. c_{p_p,p_q} is defined as the data transfer cost per GB between provider p_p and p_q. Almost every provider in current cloud market ignores the local data transfer fee in a single cloud [18]. We ignore extra fee for other services such as resource monitoring, data storage and auto-scaling workload, which are inappreciable compared with computation cost and data transfer cost [22].

The scheduler aims to minimize the workflow execution cost including computation cost and data transfer cost while satisfying its deadline constraint across multiple clouds. The whole scheduling strategy is defined as $S = (R, M, C_{total}, T_{total})$, where $R = \{vm_1, vm_2, \ldots, vm_r\}$ is a set of leased VMs from various providers, $M = \{(t_i, vm_j) | t_i \in V, vm_j \in R\}$ represents the mappings from tasks to their corresponding VMs, C_{total} is the total workflow execution cost, and T_{total} is the total workflow execution time. Fig. 3 shows a scheduling solution for the workflow displayed in Fig. 2. The workflow tasks occupy three VMs (i.e., $R = \{vm_1, vm_2, vm_3\}$), which come from two different providers (i.e., p_p and p_q). The mapping result is $M = \{(t_1, vm_1), (t_2, vm_3), (t_3, vm_2), \ldots, (t_8, vm_3)\}$. Each VM has its corresponding turn-on time T_{on} (vm_i), turn-off time T_{off} (vm_i), and VM type *type* (vm_i). A task t_i has its actual end time *AET* (t_i) after being scheduled, and the workflow execution time can be calculated after all tasks are scheduled. Therefore, the total workflow execution cost C_{total} and the total workflow execution time T_{total} are described as formula (3) and formula (4), respectively.

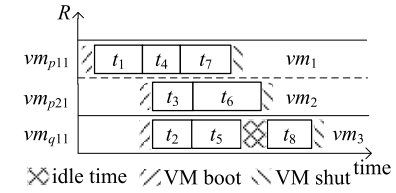

Fig. 3. A scheduling solution for the workflow displayed in Fig. 2.

$$C_{total} = \sum_{i=1}^{|R|} c_{type(vm_i)} \cdot \left\lceil \frac{T_{off}(vm_i) - T_{on}(vm_i)}{\lambda_{p(vm_i)}} \right\rceil + \sum_{j=1}^{n} \sum_{k=j+1}^{n} c_{p(t_j),p(t_k)} \cdot Data(e_{jk}) \cdot s_{jk}, \tag{3}$$

$$T_{total} = \max_{t_i \in G} \{AET(t_i)\}. \tag{4}$$

The front part of formula (3) represents the computation cost, and the rear part is the data transfer cost. $c_{type(vm_i)}$ is the vm_i cost per time unit, and p (vm_i) is the provider owning vm_i. In addition, the task t_k is the child of task t_j, and $p(t_j)$/$p(t_k)$ represent the provider executing task t_j/t_k. s_{jk} is 0 whenever both task t_j and task t_k are scheduled in the same provider or 1 otherwise.

According to the previous definitions, the optimization scheduling problem can be formally described as follows: acquire a scheduling strategy *S* with minimum C_{total} across multiple clouds while T_{total} does not exceed the deadline constraint *D*(*G*). Formula (5) shows this single-object problem with single constraint.

$$\begin{aligned} &\textbf{Minimize} \quad C_{total} \\ &\textbf{subject to} \quad T_{total} \leq D(G). \end{aligned} \tag{5}$$

Table II defines the symbols used in this paper.

IV. Proposed Strategy

In this section, we first introduce the basic particle swarm optimization algorithm, and then describe our proposed ADPSOGA algorithm in detail.

TABLE II
SYMBOLS DEFINITION

Symbol	Definition
$G = (V, E)$	A workflow with task set V and data dependency set E
t_i	Task i in G
e_{ij}	Data dependency between t_i and t_j, which indicates that t_j can not start until t_i have been completed
$D(G)$	The corresponding deadline of G
vm_{ijk}	The k^{th} launched VM with j^{th} type offered by provider p_i
$T_{exe}(t_i, vm_{ijk})$	The estimated execution time of task t_i on vm_{ijk}
$T_{boot}(vm_{ijk})$	The boot time of a new launched VM vm_{ijk}
c_{pk}	The corresponding cost per time unit for VM vm_{pkr}
$T_{intra}(e_{ij}, p_p)$	The intra-data transfer time between t_i and t_j in provider p_p
$T_{inter}(e_{ij}, p_p, p_q)$	The inter-data transfer time between provider p_p and provider p_q

A. Particle Swarm Optimization

PSO is an evolutionary computation technique inspired by the social behavior of bird flocks. It was first presented by Kennedy and Eberhart [19] in 1995 and has been broadly utilized and investigated ever since. The particle is the most important concept in PSO. A particle represents a candidate solution that can move through the optimization problem space. Each particle has its own velocity, which will determine its future direction and magnitude. This velocity is affected by the particle's personal best position *pBest* and the population's global best position *gBest*. A fitness function, $fit(X_i)$, is used to evaluate the quality of a particle X_i. Each particle is determined by its velocity and position, and they update their velocities and positions iteratively described as formula (6) and formula (7).

$$V_i^{t+1} = w \times V_i^t + c_1 r_1 \left(pBest_i^t - X_i^t\right) + c_2 r_2 \left(gBest^t - X_i^t\right), \tag{6}$$

$$X_i^{t+1} = X_i^t + V_i^{t+1}. \tag{7}$$

t is the current iteration of evolutionary population. V_i^t and X_i^t represent the velocity and position of i^{th} particle at t^{th} iteration. In general, a maximum velocity V_{max} is defined to ensure that the particle moving space is in the range of solution space. In addition, the inertia weight w determines how much the previous velocity can affect the current velocity. It has a significant impact on the algorithm's convergence. The two acceleration coefficients (i.e., c_1 and c_2) represent the particle cognitive ability to their personal best value and global best value. In order to enhance the randomness of searching space, the algorithm introduces two random numbers (i.e., r_1 and r_2) whose value is both between 0 and 1.

PSO is usually used to solve the continuous problem [23]. Workflow scheduling is a discrete problem and requires a new problem coding strategy and fitness evaluation function.

B. ADPSOGA Algorithm

The proposed ADPSOGA algorithm is illustrated from the following seven parts.

1) Problem Encoding: To improve the algorithm performance and enhance its searching efficiency, a good encoding strategy should satisfy the following three principles [33].

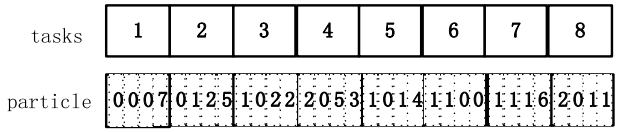

Fig. 4. A particle encoded for a workflow scheduling plan.

Definition 1 (Completeness): each candidate solution for the problem can be encoded as a particle.

Definition 2 (Viability): each particle represents a corresponding candidate solution.

Definition 3 (Non-Redundancy): there is a one-to-one correspondence between a particle and a candidate solution.

An encoding strategy satisfying the above mentioned three principles is difficult. Inspired by the work in [5], we use a provider-type-VM-order nested strategy to encode the workflow scheduling problem. A particle represents a workflow scheduling strategy, and the i^{th} particle in t^{th} iteration is shown in formula (8).

$$X_i^t = \left(x_{i1}^t, x_{i2}^t, \ldots, x_{in}^t\right), \tag{8}$$

$$x_{ik}^t = \left(p, vm_{pj}, vm_{pjr}, ord\right)_{ik}^t. \tag{9}$$

The x_{ik}^t (where $k = 1, 2, \ldots, n$) represents the k^{th} task assignment, which is described as formula (9). Therefore, the particle dimension is three times of the number of tasks. The combination (p, vm_{pj}, vm_{pjr}, *ord*) means the provider, VM type, execution VM and scheduling order of task, respectively. The value of the first three coordinates can range from 0 to its corresponding maximum number, and the value of the last coordinate ranges from 0 to n-1. When two tasks without data dependency are scheduled on the same virtual machine, the task with smaller order is executed first. The *ord* of each task in an encoded particle is different, which can guarantee the 'Non-redundancy' principle. For example, Fig. 4 shows a particle encoded for scheduling a workflow with eight tasks across multiple clouds. We assume that there are three providers and each provider has eight types of VM, so the p coordinate is from 0 to 2, and the vm_{pj} coordinate is from 0 to 7. Fig. 4 tells us that task t_1 is scheduled on vm_{000} with vm_{00} type in provider 0, and task t_2 is scheduled on vm_{012} with vm_{01} type in provider 0.

Our encoding strategy meets the 'Completeness' and 'Non-redundancy' principles. Each dimension of a particle represents the allocated location (VM) for the corresponding task. Therefore, each particle represents a candidate solution for the problem. Due to the provider-type-VM-order nested strategy, an encoded particle has only one corresponding scheduling strategy, and a scheduling strategy can only be encoded into a specific particle. However, the encoding strategy fails to satisfy the 'Viability' principle. For instance, the execution time of some workflow scheduling plans may exceed its deadline constraint.

2) Initial Resource Pool: Due to the dynamicity and elasticity of the resource supply model across multiple clouds, there is no initial resource set as an input for our strategy.

Setting an initial resource pool $R_{initial}$ too large will increase the search space and extend the convergence of PSO. It may fail to find a feasible solution if the $R_{initial}$ is too limited. Therefore, it is important to initialize an appropriate resource pool, which can not only improve the algorithm convergence but also guarantee the solution integrity.

A simple and available strategy for initializing the resource pool is to allocate one of each VM type for each task. Notice that this initialization strategy can guarantee the diversity and integrity of the algorithm. However, this strategy has large search space and increases the algorithm complexity.

In order to maintain the integrity of all possible solutions and reduce the search space, we design a suitable strategy and the size of $R_{initial}$ is $|S_{par(G)}| * Num_{type(vm)}$ (where $S_{par(G)}$ is the set including the maximum number of tasks that can be executed in parallel), and $Num_{type(vm)}$ is the amount of all VM types from multiple providers defined as follows.

$$Num_{type(vm)} = \sum_{p}^{P} Num_{vm(p)}. \tag{10}$$

$Num_{vm(p)}$ is the amount of all VM types from provider p. Therefore, this strategy makes sure that each task has the chance to select one VM of each type. In addition, we adopt the BFS [34] to calculate $S_{par(G)}$ in this study.

3) Fitness Function: The fitness function *fit*(X_i) is used to estimate the advantage and disadvantage of all the candidate solutions. In general, the fitness function with lesser value represents a better solution. Our candidate solutions may break the 'Viability' principle as described before, therefore we should consider the failed situation while defining the fitness function. We incorporate the constrained-processing strategy [35] and compare two possible solutions in three different situations.

Situation 1: If one solution is feasible and another is unfeasible, it is undisputed to select the feasible one. The *fit*(X_i) is defined as formula (11), where $T_{total(Xi)}$ is the workflow execution time of i^{th} particle.

$$fit(X_i) = \begin{cases} 0, & \text{if } T_{total(X_i)} \le D(W) \\ 1, & \text{else} \end{cases}, \tag{11}$$

Situation 2: If both solutions are unfeasible, the solution with less workflow execution time will be selected. The scheduling plan with less makespan is more likely to meet the deadline constraint.

$$fit(X_i) = T_{total(X_i)}, \tag{12}$$

Situation 3: If both solutions are feasible, it is better to select the cheaper one. The *fit*(X_i) is defined as formula (13), where $C_{total(Xi)}$ is the workflow execution cost of i^{th} particle.

$$fit(X_i) = C_{total(X_i)}. \tag{13}$$

4) Update Strategy: PSO has three main components: *inertia*, *individual cognition* and *social cognition*, which are shown in formula (6). The introduction of the crossover and the mutation operator of GA greatly enhances the search capability of PSO, which explores a wider solution space. Therefore, ADPSOGA overcome the premature convergence of traditional PSO. The new update strategy is shown in formula (14), where $M_u()$ represents the mutation operator, and $C_g()$, $C_p()$ are both the crossover operators. The *ord* of each task keeps the original value, so we only update the first three coordinates.

$$X_i^t = c_2 \oplus C_g\Big(c_1 \oplus C_p\Big(w \oplus M_u\Big(X_i^{t-1}\Big), pBest_i^{t-1}\Big), \times\ gBest^{t-1}\Big), \tag{14}$$

The *inertia* component incorporates the mutation operator of GA, and the updated *inertia* part is shown in formula (15), where r_1 is a random number between 0 and 1. $M_u()$ randomly selects an index in the particle and changes the index value irregularly but in the range of defined maximum value. Fig. 5 illustrates the mutation operator with particle shown in Fig. 4. It selects the second task as a mutation object (mp1), and the corresponding scheduling parameter changes from (0, 1, 2) to (1, 2, 0), which meets the changing criteria.

$$A_i^t = w \oplus M_u\Big(X_i^{t-1}\Big) = \begin{cases} M_u\Big(X_i^{t-1}\Big), & \text{if } r_1 < w \\ X_i^{t-1}, & \text{else} \end{cases}, \tag{15}$$

The *individual cognition* component and *social cognition* component both incorporates the crossover operator of GA, and the updated results are described formula (16) and formula (17), respectively. r_2 (or r_3) is a random number between 0 and 1. $C_p()$ (or $C_g()$) randomly selects two indexes in the mutated particle and crossovers it with the *pBest* (or *gBest*) particle between the two indexes. Fig. 6 illustrates the crossover operator for the *individual* (or *social*) *cognition* component. It randomly selects two indexes (i.e., cp1 and cp2), and replaces the old particle value between cp1 and cp2 with the *pBest* (or *gBest*) particle value.

$$\begin{aligned} B_i^t &= c_1 \oplus C_p\Big(A_i^t, pBest^{t-1}\Big) \\ &= \begin{cases} C_p\big(A_i^t, pBest^{t-1}\big) & r_2 < c_1 \\ A_i^t & else, \end{cases} \end{aligned} \tag{16}$$

$$\begin{aligned} C_i^t &= c_2 \oplus C_g\Big(B_i^t, gBest^{t-1}\Big) \\ &= \begin{cases} C_g\big(B_i^t, gBest^{t-1}\big) & r_3 < c_2 \\ B_i^t & else. \end{cases} \end{aligned} \tag{17}$$

5) A Particle to a Schedule Mapping: Algorithm I shows the pseudo-code for translating a particle position to a schedule. Its inputs are workflow *W*, initial resource pool $R_{initial}$ and particle *X*. First, it initializes four elements of the scheduling strategy *S* in line 1. After initialization, the algorithm

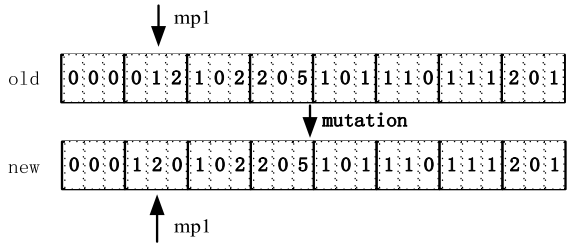

Fig. 5. Mutation operator for the *inertia* component.

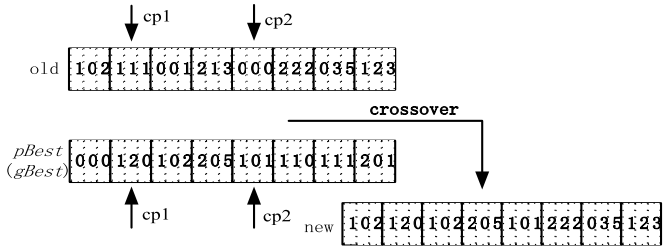

Fig. 6. Crossover operator for the *individual* (*social*) *cognition* component.

Algorithm 1 A Particle to a Schedule Mapping

procedure Schedule_Generation(*W*, R_{intial}, *X*)
1: initialize: $R \leftarrow$ *null*, $M \leftarrow$ *null*, $C_{total} \leftarrow 0$, $T_{total} \leftarrow 0$
2: calculate $T_{exe}[|W| \times |R_{intial}|]$
3: calculate $T_{intra}[|W| \times |W|]$, $T_{inter}[|W| \times |W|, |P| \times |P|]$
4: **for** $i = 0$ to $i = |W|$-1
5: $t_i = W[i], r_{X(i)} = R_{intial}[X(i)]$
6: **if** t_i has no parents **then**
7: **if** $r_{X(i)}$ does not exist **then**
8: launch $r_{X(i)}$, $LETr_{X(i)} = T_{boot}\ (r_{X(i)})$, $T_{on}(r_{X(i)}) = LETr_{X(i)}$ - $T_{boot}\ (r_{X(i)})$
9: **end if**
10: $STt_i = LETr_{X(i)}$
11: **else**
12: **call** *Max_Parents* (t_i), *maxT* = *Max_Parents*(t_i)
13: **if** $r_{X(i)}$ is not exist **then**
14: launch $r_{X(i)}$, $LETr_{X(i)} =$ **max**$(maxT, T_{boot}(r_{X(i)}))$, $T_{on}(r_{X(i)}) = LETr_{X(i)}$-$T_{boot}(r_{X(i)})$
15: **end if**
16: $STt_i =$ **max**$(maxT, LETr_{X(i)})$
17: **end if**
18: $exe = T_{exe}$[*i*][*X*(*i*)]
19: **foreach** child t_c of t_i **do**
20: **if** t_c, t_i are scheduled in the same cloud but different VMs **then**
21: $transfer+ = T_{intra}$[*i*][*c*]
22: **else if** t_c, t_i are scheduled in different clouds **then**
23: $transfer+ = T_{inter}$[*i*][*c*][*p*][*q*]
24: **end if**
25: **end for**
26: $ETt_i = STt_i + exe + transfer$
27: $M = M \cup (t_i, r_{X(i)}, STt_i, ETt_i)$
28: **if** $r_{X(i)} \notin R$ **then**
29: $R = R \cup \{r_{X(i)}\}$
30: **end if**
31: $LETr_{X(i)} = LETr_{X(i)} + exe + transfer$
32: **end for**
33: calculate C_{total} and T_{total} according to formula (3) and (4), respectively
34: output C_{total} and T_{total}
end procedure

Algorithm 2 A Task Waiting Time and Data Transfer Cost

procedure *Max_Parents*(t_i)
1: initialize: $T_{wait} \leftarrow 0$, $C_{tranfer} \leftarrow 0$
2: **foreach** parent t_p of t_i **do**
3: **if** t_p and t_i are scheduled in the same cloud but different VMs **then**
4: $T_{wait} =$ **max** ($T_{wait}, T_{intra}[p][i]$)
5: **else if** t_p and t_i are scheduled in different clouds **then**
6: $T_{wait} =$ **max** (T_{wait}, $T_{inter}[p][i][p][q]$)
7: $C_{tranfer}+= Data\ (e_{pi})*c_{q,p}$
8: **end if**
9: **end for**
10: output T_{wait}, $C_{tranfer}$
end procedure

calculates the estimated execution time of each task on every type of VM resource in line 2. This is represented as a matrix, where the columns represent the initial resources, the rows represent the tasks and the entry $T_{exe}[i][j]$ represents the estimated execution time of task t_i on VM vm_j. Also, the intra-data transfer estimated time and the inter-data transfer estimated time are calculated as matrixes in line 3. $T_{intra}[i][j]$ represents the estimated data transfer time from task t_i to task t_j in a single cloud, and $T_{inter}[i][j][p][q]$ represents the estimated data transfer time from task t_i in the cloud *p* to task t_j in the cloud *q*.

At this moment, the algorithm has the whole information for decoding the particle position and generating a candidate schedule. It iterates through each index *i* in a particle *X* and generates the *R* and *M* step by step. The algorithm firstly determines that the task t_i is scheduled on the resource $r_{X(i)}$ based on the *Problem encoding* part in line 5. The encoding strategy shows that the index *i* corresponds to the task t_i and its value *X*(*i*) corresponds to the resource $r_{X(i)}$. Then, the estimated start time STt_i of task t_i is calculated in lines 6-17. There are two scenarios for calculating the STt_i:

a) If task t_i has no parent, it can be executed as soon as the VM $r_{X(i)}$ is available. Therefore, the STt_i of task t_i is equal to the VM $r_{X(i)}$ already-leased time $LETr_{X(i)}$. In addition, if the VM $r_{X(i)}$ does not exist, it should be launched and $LETr_{X(i)}$ is equal to the booting time $T_{boot}(r_{X(i)})$.

b) If task t_i has one or more parents, it can be executed after all of its parents have been completed and the output data is transferred to VM $r_{X(i)}$. The algorithm calls *Max_Parents* (t_i) procedure to calculate the waiting time of task t_i and the data tranfer cost from its parents. In addtion, the algorithm also consider whether the VM $r_{X(i)}$ is available or not.

After obtaining the estimated start time STt_i of task t_i, the algorithm calculates the estimated end time ETt_i of task t_i based on its execution time and data transfer time in lines 18-26. There are three scenarios for calculating the data transfer time of task t_i:

a) If task t_i and its child task t_c are scheduled on the same VM, the data transfer time is 0.

b) If task t_i and its child task t_c are scheduled on different VMs but in the same cloud, the data transfer time is equal to T_{intra}[*i*][*c*].

c) If task t_i and its child task t_c are scheduled on different clouds (i.e., cloud *p* and cloud *q*), the data transfer time is equal to T_{inter}[*i*][*j*][*p*][*q*].

Finally, the new four elements of mapping result are added to the mapping set *M* in line 27. The algorithm also determines whether the VM $r_{X(i)}$ is added to the leased resource set *R* or not in lines 28-30. The newest already-leased time of VM $r_{X(i)}$ is equal to the estimated end time of task t_i in line 31. In addition, the algorithm calculates C_{total} and T_{total} according to the formula (3) and formula (4) in line 33. It outputs the final scheduling strategy *S* in line 34.

Algorithm II shows the pseudo-code for calculating a task waiting time and data transfer cost. The waiting time of a task t_i is equal to the maximum data transfer time from all its parents in lines 3-6. If a task t_i and its parent t_p are scheduled in different clouds, the data transfer cost should be considered in line 7.

6) Parameter Settings: The inertia weight w described in formula (6) determines the local (or global) searching capacity of PSO. With the smaller inertia weight w, PSO has a better local searching capacity. Otherwise, PSO has a better global searching capacity. In the initial stage, the algorithm focuses on the diversity of candidate solutions (i.e., the global searching capacity). The algorithm pays more attention to the convergence and local searching capacity in the later stage. Therefore, the value of inertia weight should be reduced with the increase of the iteration. The formula (18) is a classical tuning strategy for the inertia weight w [36]. The $w_{\max}$ is the maximum initialization value of w, and $w_{\min}$ is the minimum value of w, respectively. The $iters_{cur}$ is the current iteration, and $iters_{\max}$ is the defined maximum iteration.

$$w = w_{\max} - iters_{cur} \times \frac{w_{\max} - w_{\min}}{iters_{\max}}, \tag{18}$$

The inertia weight described in formula (18) is only related to the iteration, and it cannot meet the nonlinearity and complexity of our problem. The inertia weight actually needs to update based on the particle performance. Therefore, an adaptive tuning strategy according to the particle gap is designed for the inertia weight. A particle gap compared with the global best position at t-1$^{\text{th}}$ iteration is shown in formula (19), where $div(X^{t-1},\ gBest^{t-1})$ represents the number of different x_k^t value between X^{t-1} and $gBest^{t-1}$, and T is the number of workflow tasks. The adaptive tuning strategy for inertia weight is described in formula (20).

$$d\left(X^{t-1}\right) = \frac{div\left(X^{t-1}, gBest^{t-1}\right)}{3 \cdot T}, \tag{19}$$

$$w = w_{\max} - (w_{\max} - w_{\min}) \times \exp\left(d\left(X_i^{t-1}\right) / \left(d\left(X_i^{t-1}\right) - 1.01\right)\right). \tag{20}$$

When the value of $d(X^{t-1})$ is small (i.e., there is small gap between current particle and the global best particle), the value of inertia weight w is small and the optimized solution is searched in local area. On the other hand, the value of inertia weight w is increased and it can enhance the searching range and find the optimized solution earlier.

In addition, the other two acceleration coefficients (i.e., c_1 and c_2) are defined as formula (21) and formula (22). They are based on the linear increase (or decrease) strategy [36].

$$c_1 = c_1\_start - \frac{c_1\_start - c_1\_end}{iters_{\max}} \times iters_{cur}, \tag{21}$$

$$c_2 = c_2\_start - \frac{c_2\_start - c_2\_end}{iters_{\max}} \times iters_{cur}. \tag{22}$$

c_1*_start*, c_2*_start* are the initialization value of c_1, c_2 at the start iterative phase, and c_1*_ end*, c_2*_ end* are the initialization value of c_1, c_2 at the end iterative phase.

7) Algorithm Flowchart: Fig. 7 shows the proposed ADPSOGA algorithm flowchart and the detailed steps are summarized as follows:

Step 1: It initializes the relevant parameters such as population size, maximum iterations, inertia weight and acceleration coefficients. In addition, it generates the initialization population of particles.

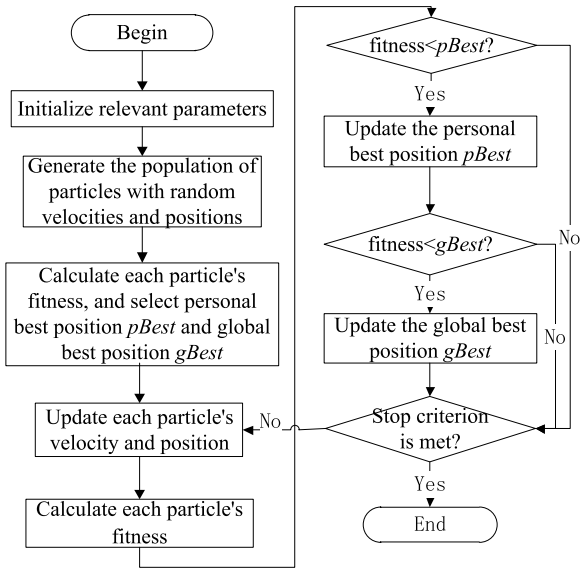

Fig. 7. ADPSOGA algorithm flowchart.

Step 2: It generates a workflow schedule according to the '*a particle to a schedule mapping*' part, and then calculates the fitness value of each particle based on the formula (11), formula (12) and formula (13). Moreover, it selects the personal best position *pBest* and the global best position *gBest*.

Step 3: It updates each particle based on the formulas in '*update strategy*' part.

Step 4: It recalculates the fitness value of each particle. If the current value of particle is better than its *pBest*, the algorithm substitutes the current value for the particle *pBest*. Otherwise, it jumps to *Step 6*.

Step 5: If the current value of particle is better than the population *gBest*, the algorithm substitutes the current value for the population *gBest*.

Step 6: It goes to *Step 3* until the stop criterion (i.e., the iteration is equal to the maximum iterations) is met.

V. Performance Evaluation

We conducted all the simulation experiments on 64-bit Win7, which is configured with 8GB memory and 2.30GHz frequency CPU. The particle number and the maximum iteration were set to 100 and 1000, respectively. We defined c_1*_start* $= 0.9$, c_1*_end* $= 0.2$, c_2*_start* $= 0.4$ and c_2*_end* $= 0.9$ based on the progressive setup [36], and these parameters were used in the rest of this work. We then described the experimental setup, the competitive algorithms of ADPSOGA and its performance results.

A. Experimental Setup

With the purpose of assessing our proposed strategy, it is better to measure it with the real-world workflows. However, there is no available library of such workflows for our experiments. We conducted our experiments with the help of five partly synthetic workflows (i.e., Epigenomics, CyberShake, Montage, SIPHT and LIGO), which were investigated in depth

TABLE III
THE BANDWIDTH BETWEEN TWO DIFFERENT CLOUDS

	Avg (MB/s)	Time/GB(s)
C1−C2	1.33	770
C1−C3	3.25	315
C2−C3	5.43	189

TABLE IV
THE DATA TRANSFER COST BETWEEN TWO DIFFERENT PROVIDERS

	Price/GB($)			
	0-1GB	1GB-1TB	1TB-10TB	10TB-50TB
C1→(C2,C3)	0.00	0.19	0.19	0.15
C2→(C1,C3)	0.12	0.12	0.12	0.10
C3→(C2,C3)	0.00	0.12	0.11	0.10

by Bharathi *et al.* [37]. They also described the basic task runtime, data transfer amount and relations between tasks in each workflow, which came from five different scientific areas such as genome sequence processing, earthquake science, astronomy, bacterial replica and gravitational physics. Therefore, the five workflows have different properties in terms of composition and components. There are four different numbers for each workflow type in a file of XML format,[1] from which we chose three sizes for the experiments in this study: Tiny (about 30 tasks), Small (about 50 tasks) and Medium (about 100 tasks) [38]. In addition, it was not 100 percent accurate to estimate the execution time of each task and we assumed that the task size was based on a normal distribution with a variation of ± 10 percent [5].

In this study, the VM performance was primarily measured by the corresponding processing capacity (i.e., CPUs) [22]. There were three clouds (i.e., EC2 (C1), Rackspace (C2), GoGrid (C3)) in our experiments, and we assumed that each cloud provided 8 VM types with specific processing capacity and cost per hour. VM processing capacity is roughly proportional to its cost, therefore we assumed that the fastest VM was coarsely 5 (8 or 10) times faster than the slowest one, and the fastest VM was coarsely 5 (8 or 10) times more expensive in the cloud C1 (C2 or C3) [27]. The cost of the cheapest VM in every cloud was the same, and each VM type had various price to performance ratio for the tasks [28]. Every one-third of tasks had better price to performance ratio in one cloud compared with the other two. According to the VM performance variations [15], the VM processing capacity was reduced at most 24 percent based on a normal distribution with a standard deviation of 10 percent and mean 12 percent.

Moreover, the intra-bandwidths in a single cloud were 20 MBps based on the analysis from Amazon services [31]. The inter-bandwidths between every two different clouds were represented in Table III, which were measured based on the iperf[2] tool. We also assumed that the data transfer variation of bandwidth was degraded at most 19 percent based on a normal distribution with a standard deviation of 5 percent and mean 9.5 percent [15]. The boot time of each VM was set to 97 seconds and the cost period was one hour [10]. The data transfer cost per GB for different clouds were shown in Table IV, which were excerpted from their own official website [27].

Finally, each workflow needs a specific deadline to verify the scheduling strategy. In order to complete the workflow within its deadline across multiple clouds, we conducted our experiments using five different deadlines as follows:

$$D_i(G) = r_i \cdot Min(G), i = \{1, \ldots, 5\}. \quad (23)$$

[1]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator
[2]http://iperf.sourceforge.net/

where $Min(G)$ was the makespan of workflow G according to the HEFT method [39], and r_i was the value selected from the deadline set $S_{deadline} = \{1.5, 2, 5, 8, 15\}$.

B. The Competitive Algorithms

As far as we know, there are no similar strategies designed for the problem described in Chapter 3. In order to verify the ADPSOGA algorithm, we modified the static MCPCPP [27] and PSO-based strategy [5] to adapt the multi-cloud environment.

Our previous work presented a workflow scheduling strategy (it was called Multi-cloud Partial Critical Paths with Pretreatment, MCPCPP), which also tried to minimize the workflow execution cost while meeting its deadline constraint across static multiple clouds. This strategy assigned sub-deadlines to the PCPs instead of independent tasks. This operation could reduce the data transfer time along the PCPs. Furthermore, it tried to find a 'best-fit instance' to execute the entire PCP before its latest finish time. This operation adapted to the new price model in commercial cloud environment. There were three important components in the MCPCPP algorithm. Firstly, in order to reduce the algorithm complexity and compress the data transfer time preliminarily, every two adjacent tasks with a common 'directed cut-edge' would be merged into a single task based on the workflow structure. Secondly, the partial critical paths with sub-deadline constraints would be found based on the 'critical parent' iterative mechanism. Thirdly, the 'best-fit' instances were allocated to each found partial critical path and all the tasks on the path were scheduled to its corresponding instance.

The MCPCPP algorithm focused on the workflow scheduling across the static multiple clouds, and it ignored that a running VM could not be shut down immediately until all the output data had been transferred to other VMs [5]. Therefore, we extended the MCPCPP algorithm with considering the boot time and shut down time of VMs in the comparative experiments.

On the other hand, the PSO-based algorithm tried to achieve the same aim as this study in a single cloud. It adopted the traditional encoding strategy based on the continuous update strategy in formula (6). The integer part of each coordinate value in a particle corresponded to a specific VM. In order to be compared with our proposed algorithm, the PSO-based algorithm was extended to adapt the multi-cloud environment with less modification (it was called Workflow Scheduling with PSO, WPSO). The overall framework of new modified WPSO algorithm was the same as the traditional PSO-based algorithm. The WPSO also determined the maximum parallel tasks in a workflow and initialized the initialization resource

TABLE V
THE AVERAGE ITERATIONS OF DIFFERENT ALGORITHMS WHEN ACHIEVING *gBest* WITH $D_1(G)$

	Algorithms	CyberShake	Epigenomics	LIGO	Montage	SIPHT
Tiny	ADPSOGA	369	392	689	612	590
	WPSO	103	273	239	174	169
	PSOGA	310	331	401	411	455
Small	ADPSOGA	688	934	605	916	511
	WPSO	159	348	212	210	213
	PSOGA	421	565	401	603	401
Medium	ADPSOGA	998	997	998	998	877
	WPSO	214	391	331	296	269
	PSOGA	553	712	781	726	581

TABLE VI
THE AVERAGE ITERATIONS OF DIFFERENT ALGORITHMS WHEN ACHIEVING *gBest* WITH $D_3(G)$

	Algorithms	CyberShake	Epigenomics	LIGO	Montage	SIPHT
Tiny	ADPSOGA	314	535	736	694	460
	WPSO	188	109	134	143	268
	PSOGA	205	385	517	428	337
Small	ADPSOGA	580	473	426	987	869
	WPSO	159	304	329	233	369
	PSOGA	310	346	378	748	547
Medium	ADPSOGA	940	938	903	998	897
	WPSO	226	346	321	298	375
	PSOGA	421	645	612	688	567

TABLE VII
THE AVERAGE ITERATIONS OF DIFFERENT ALGORITHMS WHEN ACHIEVING *gBest* WITH $D_5(G)$

	Algorithms	CyberShake	Epigenomics	LIGO	Montage	SIPHT
Tiny	ADPSOGA	426	457	332	475	695
	WPSO	252	260	184	267	134
	PSOGA	381	312	271	311	428
Small	ADPSOGA	612	585	542	762	600
	WPSO	161	322	249	257	320
	PSOGA	358	415	362	511	431
Medium	ADPSOGA	940	861	896	710	936
	WPSO	319	308	376	222	368
	PSOGA	571	612	598	577	713

pool, but this pool contained all the VM types across multiple clouds. Notice that the WPSO should consider the data transfer cost between two different clouds. The parameters setup of WPSO (i.e., inertia weight and acceleration coefficient) was the same as the PSO-based algorithm [3]. There were no modifications other than the discussions above for the competitive WPSO algorithm.

Finally, in order to test the function of '*parameter settings*' part in our proposed ADPSOGA algorithm, we adjusted the inertia weight and two acceleration coefficients in ADPSOGA according to the static definition strategy [3]. The new generated competitive algorithm (it was called Particle Swarm Optimization with Genetic Algorithm operators, PSOGA) also took into account the data transfer cost between two different clouds.

C. Performance Results

In order to test the performance of four different algorithms (i.e., MCPCPP, ADPSOGA, WPSO and PSOGA) across fluctuant multiple clouds, we generated the fluctuant factors based on their corresponding distribution functions. Each test case executed 100 times, which could promote the comprehensive assessment for each scheduling algorithm.

In addition, the five workflow types have different structures and characteristics, and it is better to normalize the workflow cost in our competitive experiments. Therefore, we introduced the normalized workflow cost *NWC*(*G*) in formula (24), where the *TEC*(*G*) was the total execution cost of workflow *G* with a specific scheduling algorithm, and the *CSC*(*G*) was the cheapest scheduling cost of workflow *G* with *cheapest scheduling* strategy [38].

$$NWC(G) = \frac{TEC(G)}{CSC(G)}. \tag{24}$$

The average iterations of different PSO-based algorithms when arriving in their corresponding *gBest* are shown from Table V to Table VII, which have different deadline constraints. With a fast view on the three tables, the iterations of three PSO-based algorithms (i.e., ADPSOGA, WPSO and PSOGA) do not decrease obviously with the weakening of deadline constraint. Note that the problem encoding and update strategy have a great influence on the iteration, and the deadline constraint has little impact on the iteration. With the increase number of tasks in a workflow, the average iterations of three algorithms when arriving in their corresponding *gBest* have been grown. This is mainly due to the population diversity, whose scale enlarges by expanding the encoding dimensions. The more number of tasks will result in more new candidate particles and larger searching space. It can increase the average iteration when arriving in the global best position. The average iteration of ADPSOGA is more than PSOGA in the same condition. It means that ADPSOGA has more population diversity than PSOGA. This is mainly due to that ADPSOGA adopted adaptive adjustment for the inertia weight and linear strategy for two acceleration coefficients. These operations result in better searching capacity and more population diversities. In addition, the PSOGA average iteration is more than WPSO obviously in the same condition. This is chiefly because WPSO adopted continuous encoding strategy to solve a discrete problem, which may fall into the premature convergence and only obtain a locally optimal solution.

Fig. 8 shows the completion rate of five medium size real-world workflows with four different scheduling algorithms (i.e., MCPCPP, ADPSOGA, WPSO and PSOGA) across fluctuant multiple clouds. Note that the 'completion' means that a workflow is completed based on a scheduling algorithm within its deadline. With a fast view on the

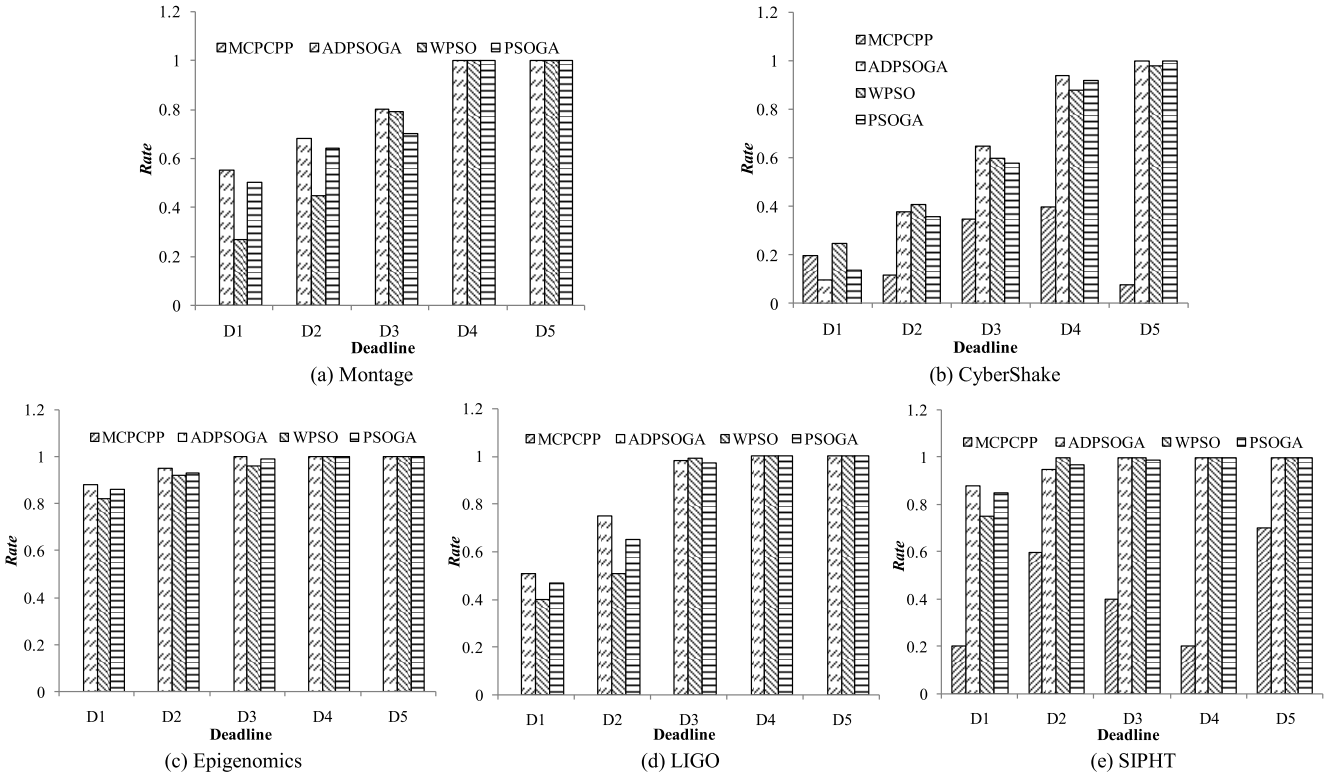

Fig. 8. The completion rate of five medium workflows with four different scheduling algorithms across multiple clouds.

five subgraphs of Fig. 8, the completion rate of three algorithms (i.e., ADPSOGA, WPSO and PSOGA) will increase with the weakening of deadline constraint. When the deadline constraint is loose (e.g., $D_5(G)$), the completion rate of these algorithms almost reach 100%. However, the completion rate of MCPCPP is obviously less than other three PSO-based algorithms and irrelevant to the deadline markedly. There is no completed case with MCPCPP algorithm in the tests for Montage, Epigenomics or LIGO workflow. Montage, Epigenomics and LIGO have more big tasks, which occupy more VM running time, compared with other two workflows. MCPCPP schedules PCP including more big tasks within its sub-deadline, which can cause more volatility in fluctuant environment.

The completion rate of Montage workflow is represented as Fig. 8(a), from which we know that the performance of ADPSOGA and PSOGA is better than WPSO. The *Rate* of WPSO is less than 30% in $D_1(G)$ phase and close to 80% in $D_3(G)$ phase. It reaches 100% in both $D_4(G)$ and $D_5(G)$ phases. Fig. 8(b) is the completion rate of CyberShake workflow. WPSO performs better than MCPCPP, and ADPSOGA has the worst performance in $D_1(G)$ phase. With the increase of deadline, the performance of ADPSOGA is slightly better than WPSO and obviously exceeds other two algorithms. The *Rate* of ADPSOGA, WPSO and PSOGA all reach 100% in $D_5(G)$ phase for the CyberShake workflow, but MCPCPP is only close to 10%. Fig. 8(c) shows the completion rate of Epigenomics workflow. The *Rate* of ADPSOGA, WPSO and PSOGA all exceed 85% in $D_1(G)$ phase, and the performance of these three algorithm is similar. The completion rate of LIGO workflow is illustrated in Fig. 8(d), from which we realize that the performance of ADPSOGA is better than other two PSO-based algorithms in both $D_1(G)$ and $D_2(G)$ phases. Fig. 8(e) shows the completion rate of SIPHT workflow. ADPSOGA is the best than any other algorithms in $D_1(G)$ phase, and PSOGA is the best in $D_2(G)$ phase. In addition, the *Rate* of three PSO-based algorithms reach 100%, and MCPCPP has a better performance with 70% completion rate.

Overall, MCPCPP has the worst performance in fluctuant environment. ADPSOGA has a better completion rate in most cases, especially with the tight deadline constraint (e.g., $D_1(G)$). This is because ADPSOGA considers the impact of VM boot time.

Fig. 9 shows the makespan of five medium size real-world workflows with four different scheduling algorithms. Three dotted lines on each subgraph correspond to three deadline constraints (i.e., $D_1(G)$, $D_3(G)$ and $D_5(G)$). The ordinate, measured in second (*sec*), represents the workflow makespan. Each workflow has different magnitudes of makespan. The Montage and CyberShake have low makespan magnitudes, and Epigenomics has the highest makespan magnitude. On the whole, MCPCPP has the smallest jump interval compared with other three algorithms. The jump interval is defined as the distance between the Q4 (upper edge) and Q0 (lower edge) of a boxplot. ADPSOGA almost has the largest jump interval except in $D_1(G)$ phase in Fig. 9(c) and Fig. 9(e). The larger jump interval means greater searching capacity in the problem space. In addition, the performance of ADPSOGA is almost better than any other algorithms except in $D_1(G)$ phase in

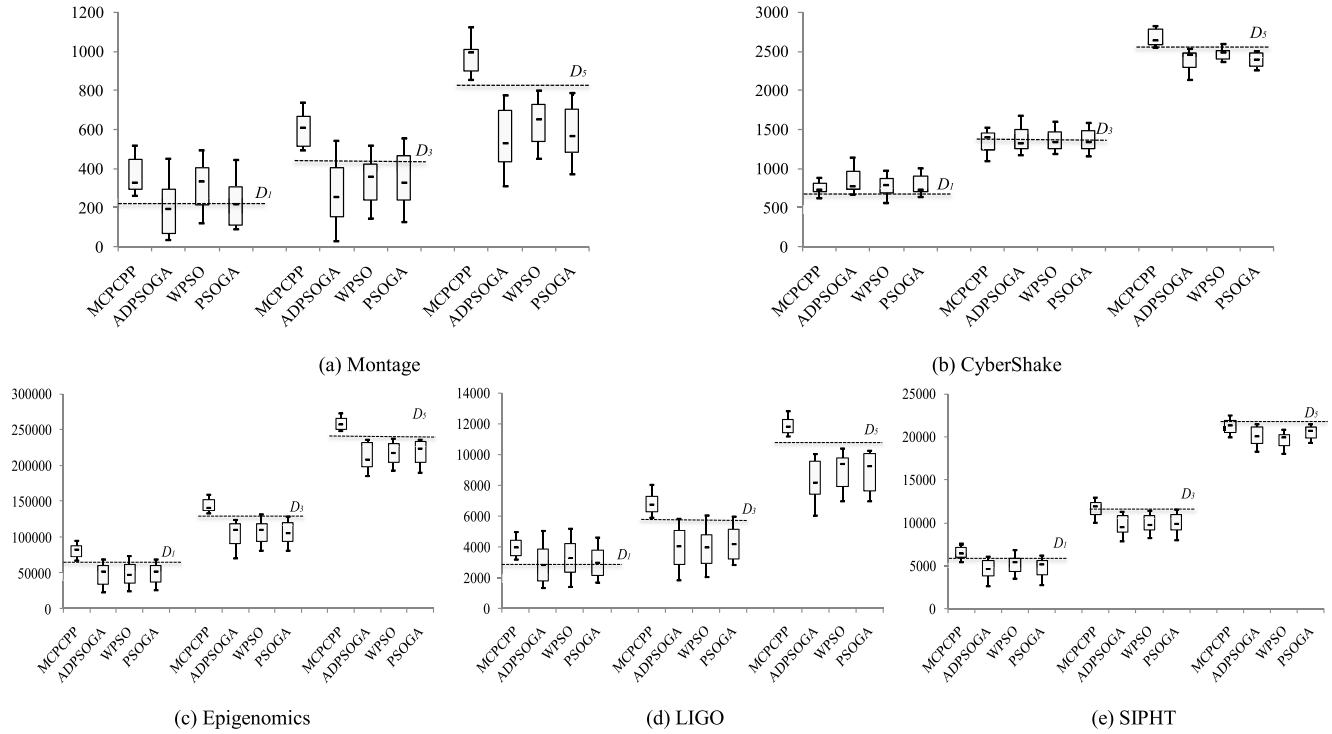

Fig. 9. The makespan boxplot of five medium workflows with three deadline constraints (i.e., $D_1(G)$, $D_3(G)$ and $D_5(G)$).

Fig. 9(b), where the Q1 (first quartile) and Q2 (median) are higher than the deadline reference line. This means that the completion rate is less than 25%.

The lower edge of MCPCPP boxplot is beyond the deadline reference line for the Montage in Fig. 9(a), Epigenomics in Fig. 9(c) and LIGO in Fig. 9(d). This means that the workflow makespan in all test cases exceed their corresponding deadline constraints. The Q2 of MCPCPP is beyond the deadline reference line in $D_3(G)$ and $D_5(G)$ phases in Fig. 9(a), which implies that more than one half test cases exceed the deadline constraints. In Fig. 9(b), the Q2 of MCPCPP is a little more than its deadline reference line in $D_3(G)$ phase, which signifies that its completion rate is less than 50%. Moreover, the reference line is between the Q2 and Q3 (third quartile) of ADPSOGA, which means that its completion rate is between 50% and 75%. In this stage, three PSO-based algorithms (i.e., ADPSOGA, WPSO and PSOGA) have similar performance for the workflow makespan, but ADPSOGA has a larger jump interval than the other two algorithms. In $D_1(G)$ phase of Fig. 9(d), the Q2 of ADPSOGA strategies to the reference line, and its completion rate is close to 50%. Also, the reference line is between Q1 and Q2 of WPSO, and the completion rate of WPSO is between 25% and 50%. In this stage, ADPSOGA has the best performance in terms of makespan and jump interval. In $D_3(G)$ phase of Fig. 9(e), the WPSO boxplot has the longest distance to the reference line compared with other three algorithms. The Q1 of ADPSOGA is similar with WPSO, but the Q4 of ADPSOGA is higher than WPSO. This phenomenon implies that ADPSOGA has larger jump interval than WPSO.

Fig. 10 shows the average makespan and average cost of five medium size real-world workflows with three deadline constraints. The reference line on each subgraph represents the corresponding deadline constraint. Both makespan and cost are displayed on the same figure, whose goal is to find a cost-efficient schedule algorithm.

For the Montage workflow, the average *NWC* of MCPCPP is the lowest compared with the other three algorithms. However, the makespan of MCPCPP exceeds its corresponding deadline in any case. This implies that MCPCPP is not a cost-efficient algorithm. In $D_1(G)$ phase, ADPSOGA and WPSO can satisfy their deadline constraints, and ADPSOGA has a lower *NWC* compared with WPSO. In $D_3(G)$ and $D_5(G)$ phases, ADPSOGA, WPSO and PSOGA can all meet their deadline constraints, and ADPSOGA has the lowest *NWC* in $D_5(G)$ phase. In addition, the average *NWC* of Montage workflow is very large, which is about 20 times more expensive than SIPHT. This is mainly because the diverse structures of different workflows. We can find that Montage has many tasks in its second level. These tasks all occupy about 15 seconds with the fastest VM type in our experiments. When the deadline is tight, the scheduling algorithm has to launch more fast VMs to ensure the second level tasks could finish within their sub-deadlines. MCPCPP generates a large makespan, which is mainly due to ignoring the VM performance variation.

For the CyberShake workflow, the average makespan of four scheduling algorithms all surpass their corresponding deadline reference lines in $D_1(G)$ phase. There is no way to assess which algorithm is better, because all the algorithms cannot satisfy their deadline constraints. In $D_3(G)$ phase, the situation is roughly similar to Montage. MCPCPP exceeds its reference line, and other three algorithms almost finish the workflow

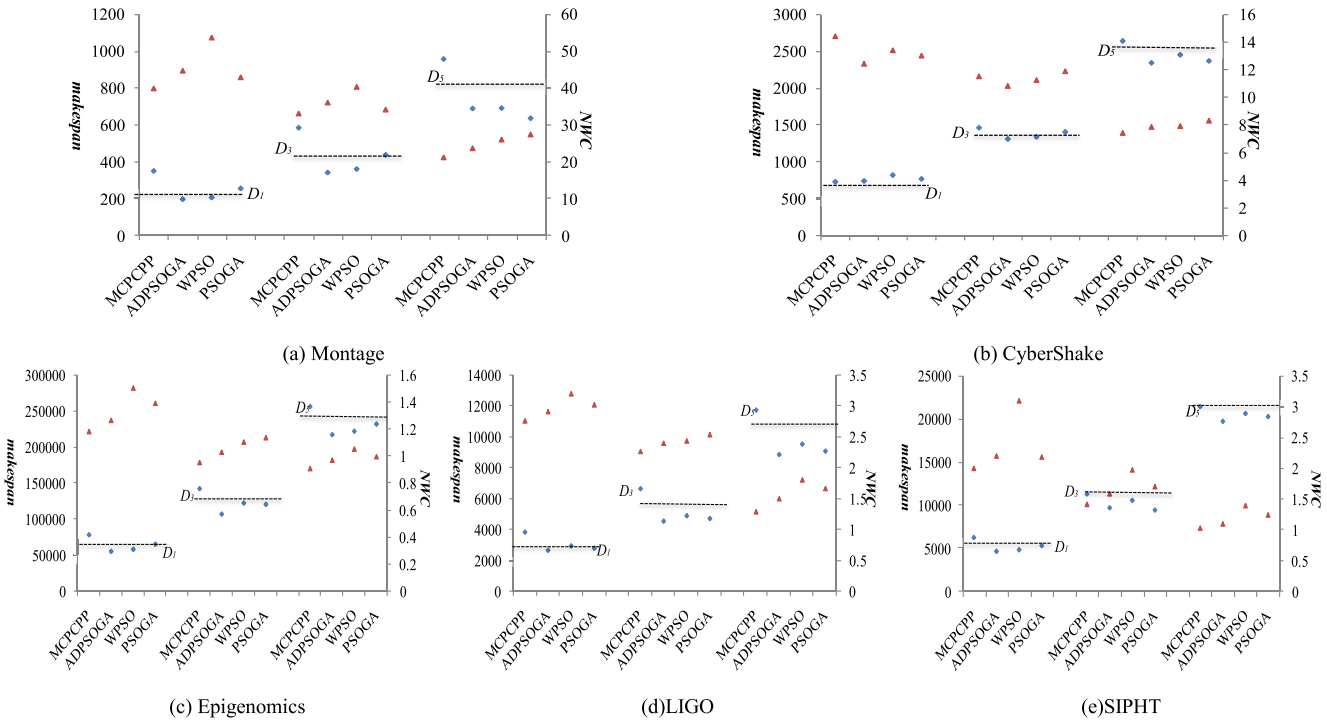

Fig. 10. The average makespan and average cost of five medium workflows with three deadline constraints (△ represents *NWC*, ◇ represents makespan).

within their deadlines. The average *NWC* of ADPSOGA is the lowest in both $D_3(G)$ and $D_5(G)$ phases.

For the Epigenomics workflow, the average *NWC* of MCPCPP is always the lowest compared with other three algorithms in any phase. However, its average makespan is obviously far more than the deadline reference line. ADPSOGA is a cost-efficient algorithm for Epigenomics in any phase. It not only has a lower average *NWC* but also meets the deadline constraint. The performance of ADPSOGA for the other two workflows (i.e., LIGO and SIPHT) is similar to its expression in Epigenomics.

In conclusion, MCPCPP in general has a lower *NWC*, but it dissatisfies its deadline constraint. Although WPSO can always meet its defined deadline, the average *NWC* is less-than-ideal. In all test cases, ADPSOGA on the whole is a cost-efficient algorithm across fluctuant multiple clouds. We do not discuss the general results for the small and tiny size workflows, because the results are similar to the medium size workflows.

Regarding the time complexity of ADPSOGA, all particles are updated and their fitness is calculated in each iteration. The number of calculations required to update the state of particles is determined by the particle number *N* and particle dimension *D*. The number of tasks *T* and the number of initial resources *R* determine the fitness function complexity based on the Algorithm I. Based on the fact that $D = 3T$ in our scheduling strategy, ADPSOGA has an overall time complexity of order $O(4N * T^2 * R)$ per iteration. In addition, the convergence time is influenced by the number of tasks and the number of initial resources.

The overall framework of WPSO and PSOGA is the same as ADPSOGA. Therefore, WPSO and PSOGA both have an overall time complexity of order $O(4N * T^2 * R)$ per iteration.

For MCPCPP, we assume that the maximum number of dependency arcs $E \sim O(T^2)$. Therefore, the time complexity of parameter initialization and complexity reduction becomes $O(T+E) \sim O(T^2)$. Then the *Schedule_all_Parents* procedure [27] schedules each task only once, and updates the parameters of its predecessors and successors. The time complexity of scheduling all tasks is O(*R***T*). In addition, each task has at most *T*–1 predecessors and successors, so its time complexity equals $O(T^2)$. Consequently, the overall time complexity of MCPCPP is $O(R * T + T^2)$.

VI. Conclusion

This study proposed an ADPSOGA strategy for scheduling a deadline-constrained scientific workflow across multiple clouds. Firstly, the strategy considered more factors such as data transfer cost, the startup/shutdown lags of virtual machines. Secondly, in order to avoid premature convergence of traditional PSO, it introduced the random two-point crossover operator and one-point mutation operator of GA. This operator could effectively improve the diversity of the population during the evolution. Finally, we designed a cost-driven strategy for the deadline-constrained workflow. It considered both the data transmission cost and the computing cost. Experimental results showed that ADPSOGA strategy had better adaptability and effectiveness across fluctuant multiple clouds.

Our strategy took more time to find the optimal solution than any other competitive algorithms. It was not suitable for dealing with the workflows that have many tasks occupying very little part of charged unit time. In the future, we will accurately determine the initial resource pool based on the

structure of workflow. It has an important influence on the scheduling efficiency and results. Another future work is to improve our algorithm to address more types of workflows and workflow ensemble across multiple clouds. Besides, we will further discuss the impact of a specific fluctuant factor on our scheduling strategy.

REFERENCES

[1] L. Liu, M. Zhang, R. Buyya, and Q. Fan, "Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing," *Concurrency Comput. Pract. Exp.*, vol. 29, no. 5, pp. 1–12, 2017.

[2] F. Marozzo, D. Talia, and P. Trunfio, "A workflow management system for scalable data mining on clouds," *IEEE Trans. Services Comput.*, vol. 11, no. 3, pp. 480–492, May/Jun. 2018.

[3] N. Xiong *et al.*, "A distributed efficient flow control scheme for multirate multicast networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 9, pp. 1254–1266, Sep. 2010.

[4] B. Lin, W. Guo, and X. Lin, "Online optimization scheduling for scientific workflows with deadline constraint on hybrid clouds," *Concurrency Comput. Pract. Exp.*, vol. 28, no. 11, pp. 3079–3095, 2016.

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

[6] H. Cao, H. Jin, X. Wu, S. Wu, and X. She, "DAGMap: Efficient and dependable scheduling of DAG workflow job in Grid," *J. Supercomput.*, vol. 51, no. 2, pp. 201–223, 2010.

[7] W.-N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 39, no. 1, pp. 29–43, Jan. 2009.

[8] H. Khajemohammadi, A. Fanian, and T. A. Gulliver, "Fast workflow scheduling for grid computing based on a multi-objective genetic algorithm," in *Proc. IEEE Pac. Rim Conf. Commun. Comput. Signal Process. (PACRIM)*, 2013, pp. 96–101.

[9] J. Jofre *et al.*, "Federation of the BonFIRE multi-cloud infrastructure with networking facilities," *Comput. Netw.*, vol. 61, pp. 184–196, Mar. 2014.

[10] J. Yin, X. Lu, C. Pu, Z. Wu, and H. Chen, "JTangCSB: A cloud service bus for cloud and enterprise application integration," *IEEE Internet Comput.*, vol. 19, no. 1, pp. 35–43, Jan./Feb. 2015.

[11] V. Varadharajan, and U. Tupakula, "Security as a service model for cloud environment," *IEEE Trans. Netw. Service Manag.*, vol. 11, no. 1, pp. 60–75, Mar. 2014.

[12] S. Bilgaiyan, S. Sagnika, and M. Das, "Workflow scheduling in cloud computing environment using cat swarm optimization," *Int. J. Comput. Appl.*, vol, 89, no. 2, pp. 11–18, 2018.

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

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

[15] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," in *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 460–471, 2010.

[16] J. Yu and R. Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," in *Proc. Workflows Support Large Scale Sci. Workshop (WORKS)*, 2006, pp. 1–10.

[17] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 8, pp. 1400–1414, Aug. 2012.

[18] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Perform. Comput. Netw. Stor. Anal.*, 2011, pp. 1–12.

[19] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. 6th IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[20] T. Sousa, A. Silva, and A. Neves, "Particle swarm based data mining algorithms for classification tasks," *Parallel Comput.*, vol. 30, nos. 5–6, pp. 767–783, 2004.

[21] J. Zhu, X. Li, R. Ruiz, and X. Xu, "Scheduling stochastic multi-stage jobs to elastic hybrid cloud resources," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1401–1415, Jun. 2018.

[22] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proc. Int. Conf. High Perform. Comput. Netw. Stor. Anal.*, 2012, pp. 1–11.

[23] Z. Wu, Z. Ni, L. Gu, and X. Liu, "A revised discrete particle swarm optimization for cloud workflow scheduling," in *Proc. IEEE Int. Conf. Comput. Intell. Security*, 2010, pp. 184–188.

[24] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, "Sky computing," *IEEE Internet Comput.*, vol. 13, no. 5, pp. 43–51, Sep./Oct. 2009.

[25] H. M. Fard, R. Prodan, and T. Fahringer, "A truthful dynamic workflow scheduling mechanism for commercial multicloud environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1203–1212, Jun. 2013.

[26] Y. Zhang, "Resource scheduling and delay analysis for workflow in wireless small cloud," *IEEE Trans. Mobile Comput.*, vol. 17, no. 3, pp. 675–687, Mar. 2018.

[27] B. Lin *et al.*, "A pretreatment workflow scheduling approach for big data applications in multicloud environment," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 581–594, Sep. 2016.

[28] G. Juve*et al.*, "Characterizing and profiling scientific workflows," *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.

[29] L. F. Bittencourt, E. R. M. Madeira, and N. L. S. Da Fonseca, "Scheduling in hybrid clouds," *IEEE Commun. Mag.*, vol. 50, no. 9, pp. 42–47, Sep. 2012.

[30] S. S. Woo and J. Mirkovic, "Optimal application allocation on multiple public clouds," *Comput. Netw.*, vol. 68, pp. 138–148, Aug. 2014.

[31] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, "Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing," *IEICE Trans. Commun.*, vol. 98, no. 1, pp. 190–200, 2015.

[32] J. L. Lucas-Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Scheduling strategies for optimal service deployment across multiple clouds," *Future Gener. Comput. Syst.*, vol. 29, no. 6, pp. 1431–1441, 2013.

[33] W. Guo, J. Li, G. Chen, Y. Niu, and C. Chen, "A PSO-optimized real-time fault-tolerant task allocation algorithm in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3236–3249, Dec. 2015.

[34] S. Jeyalatha and B. Vijayakumar, "Design and implementation of a Web structure mining algorithm using breadth first search strategy for academic search application," in *Proc. Int. Conf. Internet Technol. Secured Trans. (ICITST)*, 2011, pp. 648–654.

[35] Z. Yu, J. Zhou, W. Guo, L. Guo, and Z. Yu, "Participant selection for t-sweep k-coverage crowd sensing tasks," *World Wide Web J.*, vol. 21, no. 3, pp. 741–758, 2018.

[36] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Int. Conf. Evol. Comput. Proc. World Congr. Comput. Intell.*, 1998, pp. 69–73.

[37] S. Bharathi *et al.*, "Characterization of scientific workflows," in *Proc. Workflows Support Large Scale Sci. Workshop (WORKS)*, 2008, pp. 1–10.

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

[39] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

Wenzhong Guo received the B.S. and M.S. degrees in computer science and the Ph.D. degree in communication and information system from Fuzhou University, Fuzhou, China, in 2000, 2003, and 2010, respectively, where he is currently a Full Professor with the College of Mathematics and Computer Science. His research interests include cloud computing, mobile computing, and evolutionary computation. He currently leads the Network Computing and Intelligent Information Processing Laboratory, which is a Key Laboratory of Fujian Province, China.

Bing Lin received the B.S. and M.S. degrees in computer science and the Ph.D. degree in communication and information system from Fuzhou University, Fuzhou, China, in 2010, 2013, and 2016, respectively. He is currently an Assistant Professor with the College of Physics and Energy, Fujian Normal University. His research interest mainly includes parallel and distributed computing, computational intelligence, and data center resource management.

Guolong Chen received the B.S. and M.S. degrees in computational mathematics from Fuzhou University, Fuzhou, China, in 1987 and 1992, respectively, and the Ph.D. degree in computer science from Xi'an Jiaotong University, Xi'an, China, in 2002. He is a Full Professor with the College of Mathematics and Computer Science, Fuzhou University. His research interests include cloud computing, computation intelligence, computer networks, and information security.

Yuzhong Chen received the B.S. and Ph.D. degrees in communication and information system from the University of Science and Technology of China, in 2000 and 2005, respectively. He is currently an Associate Professor with the College of Mathematics and Computer Science, Fuzhou University. His current research interests include computational intelligence, data mining, cloud computing, and social computing. He is also the Vice Chief of the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing.

Feng Liang received the bachelor's degree in software engineering from Nanjing University in 2012. He is currently pursuing the Ph.D. degree with the Department of Computer Science, University of Hong Kong. His research interest mainly includes distributed file systems, distributed computing, and data center resource management. He is currently undertaking the research work on improving the network performance of the shuffle phase in YARN.