An Intelligent Cloud Workflow Scheduling System with Time Estimation and Adaptive Ant Colony Optimization

Ya-Hui Jia, *Student Member, IEEE*, Wei-Neng Chen, *Senior Member, IEEE*, Huaqiang Yuan, Tianlong Gu, Huaxiang Zhang, Ying Gao, and Jun Zhang, *Fellow, IEEE*

***Abstract*—The introduction of workflow in cloud computing has afforded a new and efficient way to tackle large-scale applications. As an NP-hard problem, how to schedule cloud workflows effectively and economically with deadline constraints and different kinds of tasks and resources is extraordinarily challenging. To solve this constrained problem, this paper intends to develop an intelligent scheduling system from the perspective of users to reduce expenditure of workflow, subject to the deadline and other execution constraints. A new estimation model of the task execution time is designed according to virtual machine (VM) settings in real public clouds and execution data from practical workflows. Based on the new model, an adaptive ant colony optimization algorithm is proposed to meet the quality of service and orchestrate tasks. The adaptiveness of the algorithm is embodied in two aspects. First, an adaptive solution construction method is designed that each solution is built with a dynamically changing resource pool, thus the search space of the algorithm is narrowed down and the execution time is decreased. Second, two heuristics with self-adaptive weight are introduced to adaptively meet different deadline settings. Simulating results on four types of workflows show that the proposed approach is effective and competitive.**

***Index Terms*—Ant colony optimization, cloud computing, workflow scheduling.**

I. Introduction

Workflow technology has been widely used to manage large computing applications [1]. In scientific computation environments, a workflow is defined as a collection of atomic tasks interconnected via data or computing dependencies [2], [3]. In recent decades, workflows have been applied in many fields, such as e-commerce, bioinformatics, astronomy, and physics [4]-[6]. Generally, tasks in the workflow consist of compute-intensive and data-intensive activities which should be executed in an acceptable time. To satisfy the quality of service (QoS), large-scale workflows are usually deployed and executed in distributed high-performance computing environments. How to orchestrate these tasks has been a hot topic studied for a long time [7], [8].

The appearance of the public Infrastructure as a Service (IaaS) clouds offers us a new utility-based platform to execute large scale workflows [9]-[12]. In the public IaaS model, the fundamental computing resource provided to consumers is in the form of virtual machines (VMs) which shields the underlying hardware information. Consumers can lease any number of VMs on demand and pay for what they use on the pay-as-you-go basis. In this way, the public IaaS cloud has become a popular platform for the implementation of large-scale scientific and e-commercial workflows [13], [14].

How to orchestrate workflows well is an important issue for the usage of IaaS clouds. Good orchestrations of cloud workflows can benefit service suppliers in energy saving and resources management. Meanwhile customers can also decrease the time and economic expenditure through appropriate workflow scheduling. In this paper, we consider the problem how to minimize the expenditure of executing a workflow on a public IaaS cloud under the deadline constraint from the users' perspective.

Hitherto, several researches have studied the problem in different scenarios. To reduce the expenditure, users must have an effective and economical schedule before submitting the workflow. Thus the real cloud computing model should be simulated and proper optimization methods should be used. From the perspective of the computing model, some studies directly assumed that the execution time of all tasks on all kinds of VMs is known in advance [15] which is unrealistic. A more realistic practice is to define the VM model by considering its computing capacity and price. However, either the VMs are thought to be homogeneous [16] or the capacity of a VM is represented only by the speed of central processing unit (CPU), so that the contribution of other infrastructures like memory is ignored [3], [13], [17]-[22]. Thus running a task on different VMs cost roughly the same, which is usually not the case in practice [23]-[25]. From the perspective of the scheduling algorithm, these proposed approaches can be roughly divided into two classes: heuristic and meta-heuristic. In the first

This work was supported in part by the National Natural Science Foundation of China under Grant 61622206, 61332002, and the Natural Science Foundation of Guangdong under Grant 2015A030306024. *(Corresponding Authors: Wei-Neng Chen and Jun Zhang)*

Y.-H. Jia is with Sun Yat-sen University, Guangzhou, 510006, China.

W.-N. Chen and J. Zhang are with School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510006, China and with Guangdong Provincial Key Lab of Computational Intelligence and Cyberspace Information, Guangzhou, 510006, China. (email: cwnraul634@aliyun.com; junzhang@ieee.org).

H. Yuan is with School of Computer Science and Network Security, Dongguan University of Technology, Dongguan 523808, China.

T. Gu is with the School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin, 541004, China.

H. Zhang is with the School of Information Science and Engineering, Shandong Normal University, Jinan, 250014, China.

Y. Gao is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510006, China.

category, Mao and Humphrey [26] proposed a method called Scaling Consolidation Scheduling (SCS) which used some heuristic behaviors to cut down the cost step by step. Abrishami *et al*. [3] proposed two algorithms named IaaS Cloud Partial Critical Path (IC-PCP) and IaaS Cloud Partial Critical Path with Deadline Distribution (IC-PCPD2). Although these algorithms are effective under some specific conditions, they share a common weakness that when the workflow scale becomes larger, the performance degrades rapidly owing to the lack of adaptability to different deadline requirements. Thus, several meta-heuristic approaches are proposed. Both Pandey *et al.* [17] and Rodriguez *et al.* [13] applied the particle swarm optimization (PSO) algorithm [27] in their scheduling approaches. Compared with the heuristic approaches, experimental results show that PSO performs relatively better. However, as their PSO approaches lack a mechanism to incorporate problem-based heuristic information, the search process suffers from a slow convergence speed. Considering this problem, Chen *et al.* [15] proposed an approach by applying genetic algorithm (GA), named dynamic objective genetic algorithm (DOGA). Then, in order to further accelerate the search speed, they applied the ant colony system (ACS) algorithm [28] to solve this problem, and obtained better results [29]. As a fixed encoding scheme is applied by these approaches, they all suffer from the problem of redundant search space so that the convergence speed is slow. In addition, since no effective heuristic for meeting the deadline constraint is designed in these approaches, most of them struggle to find feasible solutions when the scale increases or the constraint tightens.

Aiming at these problems, in this paper, we are going to propose an intelligent cloud workflow scheduling system which contains a more practical computing model and a more effective scheduling algorithm. The system is designed from the perspective of ordinary users. The position of the system is defined as a middleware between users and IaaS clouds. Compared with traditional scheduling systems, the proposed system makes the scheduling of cloud workflows more intelligent with respect to the following two aspects. First, speaking from the computing model, the computing capacity of VM is associated with both CPU and memory. Correspondingly, for tasks in the workflow, besides the size, each task's memory demand is also considered, and the execution time of task is estimated according to both CPU and memory. Moreover, different from the works that assume the execution time of all tasks on all kinds of VMs being known in advance, the proposed system estimates the execution time based on historical data, and thus it requires less priori knowledge. Meanwhile a feedback procedure is designed in the system to make the execution time estimation more accurate. Second, an adaptive ant colony optimization (A-ACO) method is proposed and utilized in the system to tackle the cloud workflow scheduling problem. Enlightened by the foraging behavior of ants, Dorigo *et al.* [30] first proposed the ant colony optimization (ACO) algorithm to solve the traveling salesman problem (TSP). Later, they proposed an improved ACO variant called ACS [28], which has been successfully employed for various problems [31]-[34]. The proposed A-ACO is designed on the basis of ACS. But considering the large-scale and constrained characters of the problem, some novel techniques are also designed to make A-ACO more effective and efficient. Specifically, the novelty and adaptiveness of the proposed A-ACO scheduling method are shown in two aspects:

1) By introducing an idea of dynamically changing search space, a new routing strategy and a new pheromone matrix are designed to restrict every move of each ant, so that the search space of A-ACO can be shrunk a lot. Thus A-ACO has the ability to adapt to workflows with different scales.
2) Aiming at the only constraint of the problem, i.e. deadline constraint, we have designed an effective heuristic with a dynamic weight. The proposed heuristic can directly guide ants to find feasible solutions, and the dynamic weight is able to automatically control the time when the heuristic works according to the status of the colony. Although there have been some studies which handle the constraints by changing fitness functions or weights [35], the idea of the dynamic weight in this paper is unique by adjusting the parameters of the algorithm rather than the objective function.

To demonstrate the proposed scheduling system, extensive experiments are conducted on four types of scientific workflows which consist of both data-intensive and compute-intensive tasks. Each type is tested in four different scales with three different deadline settings. Experimental results show the effectiveness and efficiency of the proposed approach.

The rest of this paper is organized as follows: in Section II, the architecture of the proposed scheduling system is shown. Section III and Section IV illustrate two important modules, the estimation module and the scheduling module, respectively. Experimental results are shown in Section V and conclusions are finally drawn in Section VI.

II. Cloud Workflow Scheduling System

Fig. 1 shows the architecture of the proposed cloud workflow scheduling system. We can see that the system consists of four modules: estimation module, scheduling module, reservation module, and execution module.

To schedule a workflow onto an IaaS cloud, seven steps should be taken:

1) At first the estimation module acquires the workflow specification and the QoS requirement from the user, and the VM specification from the cloud service provider. In line with the problem studied in this paper, the QoS here is defined as the deadline constraint.
2) Based on the acquired information, the estimation module will estimate the execution time of each task on every kinds of VM.
3) Then, taking the execution time matrix, the scheduling module will make a complete execution plan about how many VMs should be leased, when to lease and release them, and which task should be assigned onto which VM. The scheduling module consists of two components: optimization algorithm and simulation. The optimization

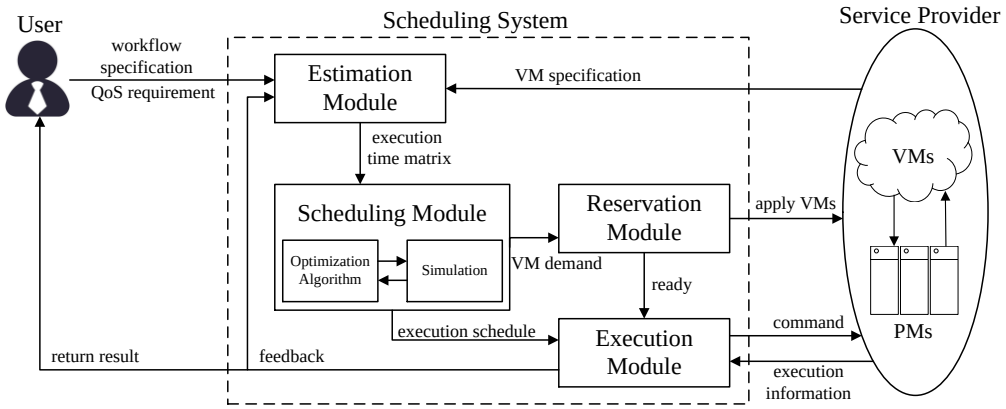

Fig. 1. Architecture of the cloud workflow scheduling system.

algorithm generates schedules and gives them to the simulation component to judge whether the schedules can satisfy the constraints and how good the schedules are. Then utilizing the simulation results, the optimization algorithm keeps finding better schedules until the stop criterion is met. Usually the stop criterion is defined as the maximum number of simulations or the execution time of the algorithm. As aforementioned, the main optimization algorithm is A-ACO. However, sometimes when the deadline constraint is very tight, A-ACO may be not capable to find feasible solutions. Under the circumstances, the Heterogeneous Earliest-Finish-Time (HEFT) [36] algorithm will be utilized to at least provide a feasible solution. If even HEFT cannot generate a feasible solution, the application will be rejected.

4) After getting a near-optimal schedule, the scheduling module tells the reservation module the amount and the types of the required VMs, and gives the execution plan to the execution module.
5) The reservation module then leases VMs from the service provider and prepares them ready.
6) When the execution module is informed that the runtime environment is prepared, it starts sending commands to the cloud to execute the workflow, telling the cloud which VM should be started or power-off.
7) During the execution, it collects the information about the real runtime of tasks and return such information to the estimation module as the feedback. If there is another similar workflow which is going to be executed, the feedback information can help in improving the precision of execution time estimation. After running the whole workflow, the execution module is also responsible for returning the final result to the user.

Among these four modules, the former two, estimation module and scheduling module, are the core of the whole scheduling system, because they together decide whether an effective and economical schedule can be generated. The latter two, reservation module and execution model, are designed to interact with cloud providers, which are all about programming, and we do not focus on them in this paper. In following sections, the estimation module and the scheduling module will be described in detail.

III. Estimation Module

Workflow scheduling on heterogeneous computing resources has been studied over the years [36], [37]. However, for the cloud workflow scheduling problem studied in this paper, new computing models are needed to characterize the workflow applications and the public IaaS clouds. Execution time estimation is conducted based on the computing model. The models about workflows and VMs are described in this section. Relative notations used in the model description are listed in Table I.

A. Workflow Specification

Workflow is represented by the task precedence graph (TPG) [33], [38], which is a directed acyclic graph (DAG) denoted as $G(V, E)$. The set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ represent the n tasks in the workflow, and each edge $e_{ij} = (v_i, v_j)$ in the edge set E means that task v_i is a direct predecessor of task v_j. In the case of scheduling workflows on clouds, every edge has a weight to represent the size of data which is transferred from the parent task to its descendant tasks. Additionally, each workflow has a deadline D defined by consumer as the constraint. A simple workflow is shown in Fig. 2.

In a workflow, we assume that each task is atomic for specific responsibility which means it cannot be further divided or interrupted during execution. According to [39], most scientific workflows are data-intensive so that the memory may influence the execution greatly. It is common that different tasks require different sizes of memory. Generally, there are an upper bound and a lower bound of memory size associated with each task. If the provided memory is smaller than the lower bound, the task is not capable to run. When the given memory is within the two bounds, the execution of the task can be

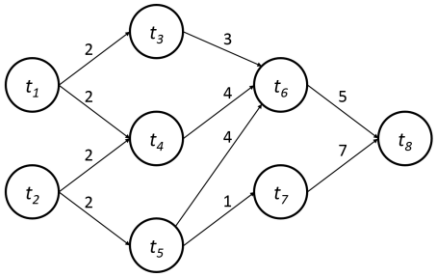

Fig. 2. A simple workflow with 8 tasks.

TABLE I
NOTATIONS USED IN MODEL DESCRIPTION AND PROBLEM DEFINITION

Notation	Meaning
i, j, k	index or counter
G	a workflow
V	set of vertices (tasks)
E	set of edges (data transfers)
v_i	the i-th task in G
e_{ij}	the edge between v_i and v_j
D	deadline
ts_i	task size of v_i
ub_i	upper bound of memory demand of task v_i
lb_i	lower bound of memory demand of task v_i
dp_i	max degree of parallelism of task v_i
su_i	speed-up of task v_i
ds_i	output data size of v_i
VM_j	the j-th virtual machine type
cn_j	capacity of virtual processors of VM_j
ms_j	memory size of VM_j
up_j	unit price of VM_j
r_k	the k-th virtual machine instance
deg_k	performance degradation of r_k
pt_i	percentage of data processing time of v_i
sc	scale of memory size to execution time
EXE_i	estimated execution time of v_i
S	a complete schedule
R	set of resources to lease
M	set of mapping relationships
TC	total cost
TT	total time
LST_k	lease start time of r_k
LET_k	lease end time of r_k
bt	boot time to initialize a VM
m_i	mapping tuple of v_i
ST_i	start time of v_i
ET_i	end time of v_i

accelerated with the growing memory until the given memory exceeds the upper bound. Such phenomenon is attributed to the fact that the growth of memory in this range can reduce the probability of page faults (hard faults), which does help in decreasing the tendency of memory swap [40]. Also different tasks have different degrees of parallelism and different speed-up ratios. Thus the attributes of task v_i (i = 1, 2, …, n) are shown as follows:

- ts_i—The task size. In this paper, we assume that the size of each task in the workflow is prior knowledge and we directly use the execution time of running the task on a VM instance which have only one processor and 1 GB memory to represent the task size [3], [13], [26].
- ub_i—The upper bound of memory demand.
- lb_i—The lower bound of memory demand.
- dp_i—The max degree of parallelism of task v_i.
- su_i—The speed-up of task v_i. It is defined as how much speed-up ratio one processor could bring. If 3 processors could reduce the execution time of v_i by half, the su_i is calculated as 2/3.
- ds_i—The output data size of task v_i.

B. Virtual Machine Model

Enlightened by the configurations of VMs in different public IaaS clouds like Amazon EC2 and Google Compute Engine, we denote each VM type VM_j with the following attributes:

- cn_j—The computing capacity of virtual CPUs in VM_j.
- ms_j—The memory size of VM_j.
- up_j—The unit price of VM_j.

In addition, as pointed out in [41], [42], the processing capacity of a VM usually degenerates more or less during the execution. Thus any VM instance, i.e. the computing resource, denoted as r_k, has a performance degradation rate deg_k. For the sake of clarity, we use r_k^j to represent that r_k belongs to the VM type VM_j. The degradation information is updated by the feedback procedure in the scheduling system.

C. Execution Time Estimation

When estimating the execution time of a task, the relationship between the execution time and the memory size is extremely hard to build accurately in practice, since the execution time of a task is related not only to the memory size and speed, but also to the memory pressure, locality, etc. [43]. Still, it is found that the miss rate of the main memory decreases when memory size grows [23]. Besides, usually the detailed hardware information is transparent to customers on a public IaaS cloud. Thus there is not a universal formula which can precisely characterize the relation among memory, CPU, and a task's execution time on cloud. But the execution time of a task can be still calculated based on the statistical running information in real applications [44], [45].

If there is not historical information that can be referred or the real test for all tasks on all kinds of VMs causes too much extra expenditure, we here can use a simple model to estimate a task's execution time based on the empirical study made by Qureshi *et al.* [46]. They found that for some data-intensive works like database task, an n-fold increase of memory size would bring roughly the same fold decrease of page faults. For some compute-intensive works, the number of page faults is independent of memory size. Thus in our model, an attribute pt_i, denoting the percentage of memory-related execution time out of the whole execution time, is utilized to divide a single task into two parts: memory-related part and CPU-related part. Separately, these two parts can be accelerated by increasing the memory size or CPU capacity. For the CPU-related part, we can use the attributes dp_i and su_i to make a rough estimation. However, for the memory-related part, the scale of memory size to execution time depends on task and hardware which needs to be measured practically [46], also such a practice is recommended by real cloud providers [47]. Thus a scale parameter between memory size and execution time, denoted as sc, is considered. Supposing that task v_i is scheduled onto VM instance r_k of type VM_j, if the upper bound of memory demand ub_i is larger than VM_j's memory size ms_j, the execution of v_i is carried out based on ms_j; otherwise ub_i is used during the execution. Overall the execution time of v_i on r_k is estimated by

$$EXE_i^{j,k} = \left(\frac{ts_i \cdot (1 - pt_i) + \dfrac{ts_i \cdot pt_i}{sc \cdot \min(ub_i, ms_j)}}{\min(cn_j, dp_i) \cdot su_i} \right) \Big/ (1 - \deg_k) . \quad (1)$$

To prove the effect of memory size and give an example of the proposed estimation method, we have conducted an experiment which is shown in the supplemental material of this paper. Without loss of generality, other estimation methods of the task execution time can be also applied in the estimation module.

Moreover, the data transfer time among VMs is omitted in the estimation module. Because usually a workflow is executed on a single data center and nowadays, data centers are built on the shared storage architecture [39], providing object storage service in which they can transfer static data by changing file index directly without costing any time. If the data transfer process is necessary to be considered, the time can be calculated according to the models shown in [13], [15].

IV. Scheduling Module

After receiving the estimated execution time of the tasks, the scheduling module will generate an effective and economical schedule. At first, the formal definitions of the objective and the constraints of the cloud workflow scheduling problem are given based on the aforementioned models. Then the schedule generation method is introduced. Finally, the specially designed A-ACO method is proposed in this section.

A. Problem Definition

In our system, the goal is to find such a schedule of a workflow, based on which the expenditure of executing the workflow on an IaaS cloud is minimized under the deadline constraint *D*. A complete schedule, represented as $S = (R, M, TC, TT)$, consists of four parts: a set of resources to lease *R*, a set of mapping relationships between tasks and resources *M*, the total cost *TC*, and the total time *TT*. $R = \{r_0, r_1, \ldots, r_{l-1}\}$ is the set of *l* VM instances, which will be used to execute tasks. It is worth noting that the size of *R* is less than or equal to the number of tasks, $l<=n$, as different tasks can be scheduled onto one VM instance. In addition, for each VM instance r_k, two lease attributes are considered when it is used: 1) lease start time LST_k, and 2) lease end time LET_k. The lease start time LST_k of a VM instance is the time when it first receives a task and the lease end time LET_k is the time when it finishes the last task scheduled onto it. Additionally the boot time to initialize a VM, denoted as *bt*, should be also considered when calculating the lease start time since most providers begin charging once the VM is power-on. $M = \{m_1, m_2, \ldots, m_n\}$ whose size is equal to the number of tasks *n*, is the mapping set with *n* tuples, maintaining allocation relationships between tasks *V* and resources *R*. Each tuple $m_i = (v_i, r_k, ST_i, ET_i)$ indicates that task v_i is arranged onto the resource r_k, is expected to start executing at time ST_i, and is estimated to complete at time ET_i.

Comprehensively, the total cost *TC* and the total execution time *TT* of a workflow are calculated by:

$$TC = \sum_{k=0}^{l-1} up_j \cdot \left\lceil \frac{LET_k - LST_k}{\tau} \right\rceil, \tag{2}$$

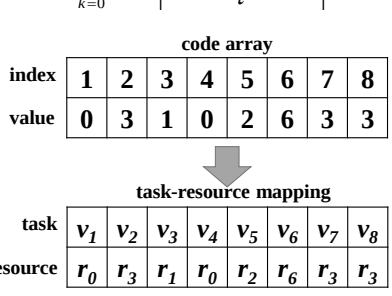

Fig. 3. A simple hypothetical array and the explanation of the array.

$$TT = \max\{ET_1, ET_2, ..., ET_n\}. \tag{3}$$

where τ is the unit time to lease a VM in IaaS, specified by service providers. Under the pay-as-you-go model, consumers pay for every unit time of the leased VM, even the unit time is partially utilized.

Accordingly, the problem studied in this paper can be formally defined as follows:

$$\begin{aligned} &\text{minimize } TC \\ &\text{subject to } TT \le D \end{aligned} \tag{4}$$

where *D* is the deadline of the workflow.

B. Encoding

Though theoretically the amount of resources in an IaaS cloud is infinite, to define the search domain of the proposed approach, an upper limit of available resources is calculated as in [13] by multiplying the maximum number of tasks that can be executed in parallel with the number of available VM types:

$$|AR| = p \cdot q, \tag{5}$$

where *p* is the maximum number of tasks that can run in parallel and *q* is the number of VM types. Under such a search domain setting, the maximum number of instances of each VM type is *p*, since there are at most *p* tasks running at the same time. Before scheduling, these available resources are indexed from 0 to $|AR|$-1. The instances which share a same VM type are ordered consecutively. Taking the workflow in Fig.2 as example, the maximum number of tasks that can be executed in parallel is 3: $\{t_3, t_4, t_5\}$ or $\{t_3, t_4, t_7\}$. Suppose that there are 3 VM types VM_0, VM_1, and VM_2, then for this workflow, 3·3=9 VM instances are available to lease. These 9 VM instances are ordered from 0 to 8: $\{r_0, r_1, r_2\}$ belonging to VM_0, $\{r_3, r_4, r_5\}$ belonging to VM_1, and $\{r_6, r_7, r_8\}$ belonging to VM_2.

When meta-heuristic algorithms are employed to solve real problems, the basic prerequisite is to encode the problem properly. Observing the schedule *S*, we can find that the most important part in *S* is the mapping relationship *M* between tasks and computing resources, because the rest of *S* can be derived from *M*. This observation inspires us to encode the set *M* to represent the solution of the problem. In the algorithm, an integer array with *n* elements $arr[1\ldots n]$ is used to represent the mapping set *M*, the index *i* of which denotes task v_i and the corresponding value $arr[i]$ represents resource $r_{arr[i]}$ that v_i is scheduled onto. A simple solution code is depicted in Fig. 3. In the example, the value of each element is within [0, 8] and the 7th value of the array is 3, indicating that task v_7 is scheduled onto resource r_3.

Although we get infinite resources in an IaaS cloud, the code scheme designed in this paper still allows the occurrence that some tasks wait for some occupied resources to release. Because under the pay-as-you-go basis, sometimes, different tasks using a same resource can make full use of the resource's lease time, so that the total cost may decrease.

C. Decoding

After encoding the mapping set *M*, a decoder is designed to translate it into a complete schedule. The pseudo code of the decoder is depicted in Algorithm 1.

To begin with, the set of resources R and the mapping set M are initialized empty and the total cost TC and the total time TT are set to 0 (Line 1). Then, the decoder deals with the tasks one by one in order (Line 2). Suppose that the decoder is arranging task v_i. First, obtaining the index of the allocated resource for v_i, we check whether resource $r_{arr[i]}$ is already leased. Two contrary cases are considered:

1) If resource $r_{arr[i]}$ is already in R which means this resource has been used by some other tasks before, we do not need to lease it again (Line 3). Then we find all parents of task v_i (Line 4). If v_i has no parent, it starts running immediately once the allocated resource $r_{arr[i]}$ is free (Line 5). Otherwise, it will wait until two conditions are satisfied: a) all its parents finish running; b) the resource $r_{arr[i]}$ is free (Line 6-9). The maximum end time of the parents is denoted as $maxET$.
2) Otherwise, we should lease $r_{arr[i]}$ at first. In the decoder, it is accomplished by instantiating a new VM instance and adding it into R (Line 11-12). Note that the VM type of $r_{arr[i]}$ can be obtained through $\lfloor arr[i]/p \rfloor$, because all the available resources are indexed in the order of VM type. Subsequently all parents of v_i are found (Line 13). If no parent exists, v_i begins to run when the corresponding VM instance $r_{arr[i]}$ boots up and the lease start time of $r_{arr[i]}$ is set to 0 (Line 14-15). Otherwise task v_i starts when its all parent tasks finish and the lease start time of $r_{arr[i]}$ is ST_i-bt where ST_i is the start time of v_i (Line 16-21).

Once the start time of v_i is determined, the execution time EXE_i can be obtained according to (1) (Line 22). Then the end time of v_i can be calculated by $ET_i=ST_i+EXE_i$. As for $r_{arr[i]}$, its lease end time is updated to the end time of task v_i, $LET_{arr[i]}=ET_i$ (Line 23-24).

Finally when all values of the elements in a mapping tuple m_i are obtained, we add m_i into M (Line 25-26). All the above procedures continue until all tasks are scheduled. Then the total cost TC and the total time TT of the workflow under such arrangement are calculated according to (2) and (3) (Line 28). Eventually an integrated schedule S is generated (Line 29).

Algorithm 1. Decoder

Input: workflow G; VM types VM; code array $arr[n]$; maximum parallel tasks number p;
Output: A schedule S;

Starts
01 Initialization: $R=\emptyset$, $M=\emptyset$, $TT=0$, $TC=0$;
02 **for** i = 1 to n
03 **if** R contains $r_{arr[i]}$ then
04 Get all parents of v_i, denoted as par_i;
05 **if** $par_i == \emptyset$ then $ST_i = LET_{arr[i]}$;
06 **else**
07 $maxET = \max\{ET_k \mid v_k \in par_i\}$;
08 $ST_i = \max(LET_{arr[i]}, maxET)$;
09 **end if-else**
10 **else**
11 Instantiate $r_{arr[i]}$ with type $VM_{arr[i]/p}$;
12 $R = R \cup \{r_{arr[i]}\}$;
13 Get all parents of v_i, denoted as par_i;
14 **if** $par_i == \emptyset$ then
15 $ST_i = bt$; $LST_{arr[i]} = 0$;
16 **else**
17 $maxET = \max\{ET_k \mid v_k \in par_i\}$;
18 $ST_i = maxET$;
19 $LST_{arr[i]} = ST_i - bt$;
20 **end if-else**
21 **end if-else**
22 calculate EXE_i according to (1);
23 $ET_i = ST_i + EXE_i$;
24 $LET_{arr[i]} = ET_i$;
25 $m_i = (v_i, r_{arr[i]}, ST_i, ET_i)$;
26 $M = M \cup \{m_i\}$;
27 **end for**
28 calculate TC and TT according to (2) and (3);
29 $S = (R, M, TC, TT)$;
Ends

D. Adaptive Ant Colony Optimization Approach

As presented in (4), the problem studied in this paper is a constrained optimization problem. To deal with such a problem, a feasibility-based rule to pairwise compare individuals proposed by Deb [48] is incorporated into the A-ACO approach. Whenever two solutions compete with each other, three rules are applied to determine which one is better: 1) any feasible solution is better than any infeasible solution; 2) between two feasible solutions, the one with better objective function value is preferred; and 3) between two infeasible solutions, the one with smaller degree of constraint violation is better.

1) Overview of the ACO Algorithm

ACO was first developed by Dorigo *et al.* [30], [49]. The underlying idea of ACO is to simulate the routing method of ants. Generally an ACO algorithm is composed of two procedures:

1. Solution Construction—During each iteration, a set of artificial ants construct solutions by selecting solution elements step by step in a finite set of available solution elements, guided by a stochastic mechanism. Components with more pheromone are more attractive to ants.
2. Pheromone Updating—The pheromone update procedure is used to increase the pheromone values associated with good solutions, and to decrease those with poor ones.

Through introducing different pheromone management methods, various ACO variants have been developed [30], such as ant system (AS) [50], ant colony system (ACS) [28], max-min ant system (MMAS) [51], etc. In this paper, the proposed A-ACO is developed based on ACS.

Two hallmarks show the differences between ACS and other ACO algorithms: 1) the pseudo random proportional rule applied in the solution construction procedure. Under this state transition rule, the artificial ants, with a certain probability, will directly choose the solution components associated with the maximum product of pheromone and heuristic values, rather than playing the roulette wheel selection strategy. Consequently, ACS takes full advantage of the past search experience of ants and obtains a fast convergence speed; 2) two pheromone updating rules, namely global updating and local updating. The former one only allows the global best ant to

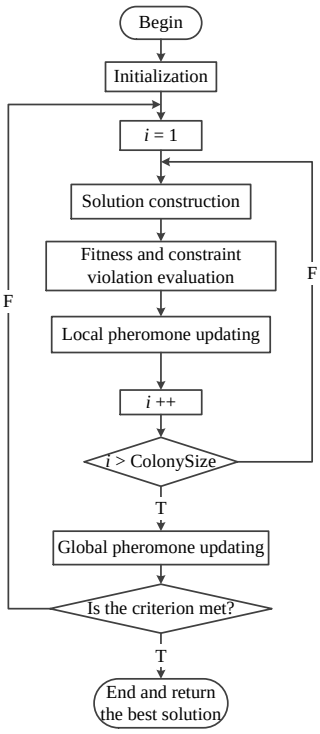

Fig. 4. Flowchart of the proposed A-ACO.

deposit pheromone, which is beneficial for fully mining the neighborhood of the global best solution. On the contrary, local updating rule is used to shuffle the tours. Every ant decreases the pheromone values in its tour instead of depositing pheromones in order to make the edges traveled in current iteration less desirable in the next iteration. In this way, the diversity of the search is kept.

These two advantages of ACS give rise to its usage in the following proposed A-ACO in this paper.

2) Adaptive Ant Colony Optimization (A-ACO)

The overall flowchart of the proposed A-ACO is shown in Fig. 4. There are five main components: initialization, solution construction, fitness and constraint violation evaluation, local pheromone updating, and global pheromone updating. Following, these five components are introduced in detail.

a) *Initialization*

In the initialization phase, the pheromone matrix is initialized and the heuristic information is calculated in order to make preparation for the solution construction process.

Pheromone. Assume there are n tasks in the workflow and the maximum number of available resources is $|AR|$. Then we can maintain a $n \times |AR|$ pheromone matrix ***PH*** with element $\varphi(i,k)$ denoting the pheromone value of arranging tasks v_i onto resource r_k.

In the ACS algorithm, the initial pheromone value φ_0 is problem-dependent. In our work, it is set as

$$\varphi_0 = \frac{1.0}{TC_{heft} \cdot n} \cdot \frac{TT_{heft}}{D}, \quad (6)$$

where TC_{heft} is the total cost and TT_{heft} is the total execution time, both produced by the Heterogeneous Earliest-Finish-Time (HEFT) algorithm [36] which arranges every task onto its most "suitable" resource; D is the predefined deadline. The concept "suitable" is defined by the following heuristic.

Heuristic. When ants select VMs for tasks, not only the pheromone contributes, but also the associated heuristics should make difference. In this paper, to accelerate the convergence speed of A-ACO, two kinds of heuristic, which are designed to directly cater to the objective in (6), i.e. *TC* and *TT*, are applied in the solution construction process.

The first one is the cost heuristic, which describes the cost of running task v_i on VMs with type VM_j, denoted as $cost_{i,j}$. Since our objective is to minimize the expenditure, a relatively smaller value of $cost_{i,j}$ implies that the resources with the type VM_j are more *suitable* for task v_i than others. It should be noted that in the initialization process, we have not leased any VM instance yet. Thus we can only measure whether a VM type, rather than a specific VM instance, is suitable for a task. The $cost_{i,j}$ is estimated by

$$cost_{i,j} = \left(\frac{ts_i \cdot (1 - pt_i) + \dfrac{ts_i \cdot pt_i}{sc \cdot \min(ub_i, ms_j)}}{\min(cn_j, dp_i) \cdot su_i} \right) \cdot up_j . \quad (7)$$

The second heuristic information applied in the approach is a dynamic value calculated during the solution construction. In an IaaS cloud with enough computing resources, if a schedule cannot fulfill the deadline constraint, it is highly likely that many tasks are assigned onto a same VM instance. Thus, the second heuristic information is designed based on the number of tasks that has been assigned to a VM, denoted as nta_k. During the solution construction, if there are already n_k tasks assigned to r_k, then nta_k is calculated by

$$nta_k = \frac{n - n_k}{n} \quad (8)$$

where n is the total number of all tasks.

Combined the two aforementioned heuristics together, the heuristic for task v_i choosing resource r_k with type VM_j (denoted as $\eta(i,k)$) is given by

$$\eta(i,k) = \frac{(nta_k)^{\alpha}}{(cost_{i,j})^{\beta}}, \quad (9)$$

where α, called the governor, is used to sense the stage of the algorithm and correspondingly adjust the weight of the *nta* heuristic. It is associated with the number of infeasible solutions. For example, suppose there are 100 ants in the colony. At the i-th iteration, 25 ants build feasible solutions and 75 ants build infeasible solutions. Then at the (i+1)-th iteration, α=75. Through the governor, the algorithm can adjust its preferences to different resources adaptively.

If the deadline constraint is loose, most of the ants will be able to construct feasible solutions. In such situation, we do not need to lease too much VMs and the governor α will be always close or equal to 0, making the *nta* heuristic contribute little to the selection. If the deadline constraint is tight, most solutions found will be infeasible at the early stage of the algorithm. At this point, we want the algorithm to find feasible solutions rather than optimizing the total cost. Meanwhile, α is close to

the colony size, making the *nta* heuristic play a key role in the resource selection. Attracted by it, the artificial ants tend to choose new resources at this stage rather than use the already leased VMs. At the later stage of the algorithm, when feasible solutions are found, α decreases and the *nta* heuristic gives way to the *cost* heuristic. Then, the algorithm truly begins to optimize the objective rather than find feasible solutions. Overall, combining the feasibility-based rule and the dynamic weight of *nta*, A-ACO is able to not only change objectives during the optimization process but also change the corresponding heuristics adaptively.

b) *Solution Construction*

In the A-ACO algorithm, we have designed a novel solution construction method to shrink the search space of the problem thus decreasing the time complexity of the algorithm. It is shown in Algorithm 2. This construction method is designed based on two facts: a) when we arrange a task, unleased resources with the same VM type are actually identical; b) once a resource is leased to run a task, it differs from other resources as it owns a unique computing ability degradation (*deg*) and lease start/end time (*LST*/*LET*). Thus it is unnecessary to consider every available resource in *AR* for every single task. To better understand the solution construction process, the example shown in Fig. 5(a), which corresponds to Fig. 3, is taken as an explanation: originally, there are 9 instances in *AR*. At first, three VM instances r_0, r_3, r_6 whose indexes are minimum in their type are put into the option set *os* (Step b). According to the first fact, $\{r_1, r_2, r_4, r_5, r_7, r_8\}$ will not be considered. Then A-ACO selects one resource for task v_1 according to the pseudo random proportional rule [28] (Step c_1-c_3). Suppose resource r_0 is selected (Step c_4). According to the second fact, now r_0 is different from r_1 and r_2. Then, we add the next several VMs into the option set (Step c_5). The number of the added VMs is an important parameter of A-ACO, which is denoted as *AD*. In the example, *AD* is set to 1, thus r_1 is added.

Algorithm 2. Solution Construction

Input: pheromone matrix ***PH***; heuristics $\eta(i, k)$; available resources *AR*

Output: an array code *arr*[*n*];

Auxiliary Storage: option set, denoted as *os*;

step a: $os = \emptyset$, $arr[n] = \emptyset$;

step b: Initialize *os* by including the first instance of each VM type;

step c: for $i = 1$ to n

step c_1: randomly generate a number $0<X<1$ and compare X with a threshold value X_0;

step c_2: if $X<X_0$, the resource r_k from the *os* with the largest value of $\varphi(i, k)\cdot\eta(i, k)$is selected.

step c_3: Otherwise, the resource is chosen through roulette wheel strategy. The probability of choosing r_k is calculated by

$$P(i,k)=\begin{cases}\dfrac{\varphi(i,k)\cdot\eta(i,k)}{\sum_{r_u\in os}\varphi(i,u)\cdot\eta(i,u)}, & \text{if } r_k\in os,\\ 0 & \text{otherwise.}\end{cases} \quad (10)$$

step c_4: $arr[i] = k$;

step c_5: $os=os\cup\{r_{k+1}, r_{k+2},\ldots\ldots, r_{k+AD}\}$;

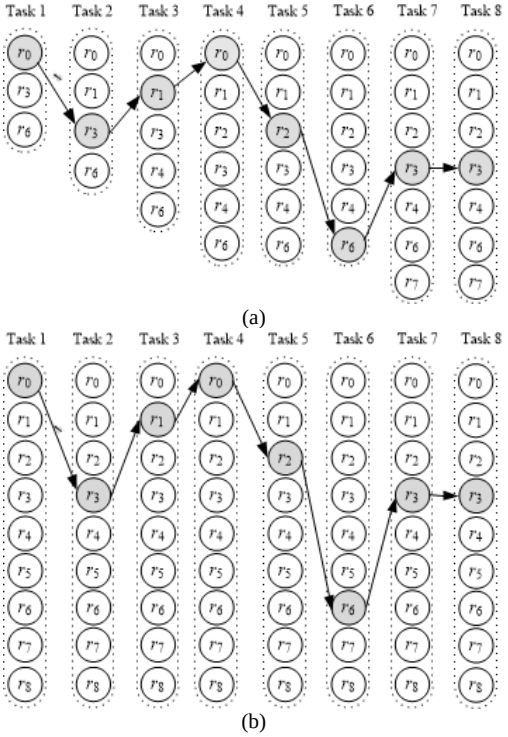

Fig. 5. Solution construction methods. (a) A-ACO. (b) Traditional ACO.

Next, A-ACO selects resources for the other tasks one by one. After allocating task v_4, we can find that r_1 is already in *os*, so we keep *os* unchanged. When all tasks are allocated, one solution *arr* is generated. What calls for special attention is that all selection behaviors are taken under the lower memory bound constraint that a task cannot be assigned to a VM which has smaller memory size than the task demands.

Since the optional components in each step are limited in the option set *os* whose size is always smaller than or equal to $|AR|$, the search space of the A-ACO is always smaller than $n^{|AR|}$. To compare the search spaces of A-ACO and the traditional ACO method, the construction method of traditional ACO is shown in Fig. 5(b). The parameter *AD* is set to accommodate the ACS algorithm, and we will investigate this parameter sufficiently in the experiment section.

c) *Fitness and Constraint Violation Evaluation*

In this scheduling problem, the total cost *TC* is considered as the fitness of a solution and whether a solution violates the deadline constraint is determined according to the total time *TT*. If the total time *TT* is less than or equal to the pre-defined deadline *D*, the corresponding solution is feasible. Otherwise the solution is infeasible. Between two infeasible solutions, the one with smaller *TT* is better than the other.

d) *Pheromone Updating*

Whenever an ant builds a solution, the algorithm updates pheromone values according to the local updating rule. After all ants finish their solution constructions, pheromones are further updated according to the global updating rule. These two rules share a same updating formula symbolically which is

$$\varphi(i,k)=(1-\rho)\cdot\varphi(i,k)+\rho\cdot\Delta\varphi(i,k). \quad (11)$$

The difference between these two rules is the value of $\Delta\varphi(i, k)$.

Global Updating. In the global updating rule, $\Delta\varphi(i, k)$is calculated by

TABLE II
NUMBER OF TASKS IN EACH WORKFLOW

Name	small	medium	large	xlarge
Montage	25	50	100	1000
CyberShake	30	50	100	1000
LIGO	30	50	100	1000
SIPHT	30	60	100	1000

$$\Delta\varphi(i,k)=\begin{cases}1/TC_{gbest}, & \text{if } arr_{gbest}[i]==k\\ 0, & \text{otherwise}\end{cases}, \quad (12)$$

where *gbest* is the global best solution found so far.

Local Updating. In the local updating rule, $\Delta\varphi(i, k)$ is calculated by

$$\Delta\varphi(i,k)=\varphi_0,\ arr[i]==\text{k}, \quad (13)$$

where φ_0 is the initial pheromone value calculated by (6).

Overall, through the collaboration of the adaptive *nta* heuristic and the pairwise comparison method [48], the proposed A-ACO gains a fast speed to enter the feasible zone of the search space. Meanwhile through the collaboration of the *cost* heuristic and the novel solution construction method, A-ACO is capable to select suitable resources for different tasks properly.

V. EXPERIMENTAL STUDIES

A. Experimental Settings

To verify the feasibility and efficiency of the proposed model and A-ACO, four different types of workflows which have been widely applied in this domain are used in the experiments: Montage, CyberShake, LIGO, and SIPHT [52]. Montage was created by the NASA/IPAC Infrared Science Archive to generate custom mosaics of the sky. The CyberShake workflow is used to characterize earthquake hazards, in which most of tasks can be seen to be compute-intensive. The LIGO Inspiral Analysis is used to analyze the gravitational wave data produced by various events in the universe. The SIPHT is an application about bioinformatics which uses a workflow to automate the search process for sRNA encoding-genes for all bacterial replicons in a specific database. Details about these four kinds of workflows can be found in [52]. For each kind of workflow, four different scales are considered in the experiments. The number of tasks in each workflow is shown in Table II.

As for the computing resources, all VM instances considered in this paper simulate from the Amazon EC2 on-demand instances. Three types of VMs are adopted: general-purpose, compute-optimized, and memory-optimized. Moreover, there are four or five configurations in each type. Since the processing capacity of each VM is not linearly proportional to the number of virtual CPUs in the Amazon EC2, we apply the ECU which is a unified measurement of processing capacity to represent the number of CPU. Details about VM configurations and prices are shown in Table III. Additionally, the unit time to lease a VM instance τ in the Amazon EC2 is 1 hour.

TABLE III
CONFIGURATIONS AND PRICES OF DIFFERENT VMS

Type	Name	ECU	Memory(GB)	Price($/hour)
General Purpose	m3.medium	3	3.75	0.070
	m3.large	6.5	7.5	0.140
	m3.xlarge	13	15	0.280
	m3.2xlarge	26	30	0.560
Compute Optimized	c3.large	7	3.75	0.105
	c3.xlarge	14	7.5	0.210
	c3.2xlarge	28	15	0.420
	c3.4xlarge	55	30	0.840
	c3.8xlarge	108	60	1.680
Memory Optimized	r3.large	6.5	15	0.175
	r3.xlarge	13	30.5	0.350
	r3.2xlarge	26	61	0.700
	r3.4xlarge	52	122	1.400

TABLE IV
ALL WORKFLOWS TESTED IN THE EXPERIMENT

Kind	Name			
Montage	M_25_1	M_50_1	M_100_1	M_1000_1
	M_25_2	M_50_2	M_100_2	M_1000_2
	M_25_3	M_50_3	M_100_3	M_1000_3
CyberShake	C_30_1	C_50_1	C_100_1	C_1000_1
	C_30_2	C_50_2	C_100_2	C_1000_2
	C_30_3	C_50_3	C_100_3	C_1000_3
LIGO	L_30_1	L_50_1	L_100_1	L_1000_1
	L_30_2	L_50_2	L_100_2	L_1000_2
	L_30_3	L_50_3	L_100_3	L_1000_3
SIPHT	S_30_1	S_60_1	S_100_1	S_1000_1
	S_30_2	S_60_2	S_100_2	S_1000_2
	S_30_3	S_60_3	S_100_3	S_1000_3

In order to conduct the simulation, for each task, a percentage of memory-related execution time *pt*, an upper bound of memory demand *ub* and a lower bound of memory demand *lb* are generated according to their own properties as described in [52]. Before generating a specific number of *pt*, we classify the tasks into three categories: compute-intensive, general, and data-intensive. For these three kinds of tasks, *pt* is randomly generated within (0, 0.3], (0.3, 0.7], and (0.7, 1.0) respectively. Referring to the two bounds of memory demand, for small tasks, *ub* is randomly generated within [1.0, 7.5) and *lb* is set to 0; while for big tasks, *ub* is in the range of [7.5, 30) and *lb* is in the range of [1.0, 7.5). The range [30, 120) of *ub* is specially prepared for the extraordinarily large tasks, which require vast size of memory and their *lb* values are generated within [7.5, 30). It should be mentioned that all these random values follow the uniform distribution and are generated only once for all algorithms. For real applications, we recommend readers to check the peak memory usage, page fault rate, data throughput of tasks before deciding the aforementioned parameters based on the methods and measurements used in [14], [23], [52].

Additionally, the boot time of each VM instance in the experiments is set to 97 seconds [26]. The degradation of the processing capacity of a VM instance follows a normal distribution N(0.12, 0.10) with maximum value of 0.24 [42]. Moreover, to facilitate the experiments, the scale of memory size to execution time *sc* is set to 1.

To verify the adaptability of the proposed approach to different deadline settings, three different deadlines are generated according to:

$$D = fastest + (slowest - fastest)\cdot\frac{3\cdot fastest}{\Delta t\cdot slowest}, \quad (14)$$

where *slowest* is the execution time of the workflow obtained by mapping each task onto the cheapest VM while satisfying the lower bound of memory demand; *fastest* is the execution

time of the workflow obtained by allocating each task to a VM instance of the most expensive VM type. In addition, Δt is set to 1, 2, and 3 for the three deadlines, respectively. The larger Δt is, the stricter the deadline is. What calls for special attention is that the period *slowest–fastest* is not trisected in (15), because for some workflows, the value of *slowest* is two orders of magnitude higher than the *fastest*. If the period *slowest–fastest* is trisected, even the strictest deadline:

$$D = fastest + (slowest - fastest)/3 \quad (15)$$

is very easy for most approaches to reach. Thus in (14), the multiple relationship between the *fastest* and the *slowest* is considered to make a fair deadline setting.

On the basis of the above description, a nomenclature "name_size_deadline" is used to identify each specific workflow. All workflows tested in the experiment are listed in Table IV, such as M_25_1 refers to the Montage workflow with total 25 tasks and the most relaxed deadline.

To testify the efficiency of the proposed A-ACO, we compare it with two other representative meta-heuristic methods, i.e. the PSO method [13] and the ACS [29] method. The PSO method [13], [17] can be taken as the first evolutionary algorithm which is used to the problem studied in this paper, and it has outperformed some well-known heuristic methods according to their experimental results. The ACS method [29] was proposed recently which has achieved very good performance. To make these two approaches adapt to the situation described in this paper, the encoding and decoding schemes proposed in Section III are embedded into these algorithms. In addition, the HEFT algorithm is tested to make a baseline of the performance. Since the HEFT method tends to assign new VMs to tasks, in most cases it will get feasible solutions as tasks will never wait for an occupied VM.

For the parameters in A-ACO, we directly use the settings in canonical ACS [28], X_0=0.9 and ρ=0.1. β is set to 5. The colony size is set to 10 and the number of generation is 500. With same colony size and generation number, other parameters of the compared ACS method are set according to [29]. As for PSO, its parameters are set as recommended in [13] with 100 particles and 250 generations. Additionally, 20 independent runs for each algorithm on every workflow are conducted, based on the simulation tool CloudSim [53]. Besides, although HEFT is a deterministic algorithm, due to the randomness of the VM's degradation on processing capacity, its results in different runs are also slightly different, especially for some large workflows.

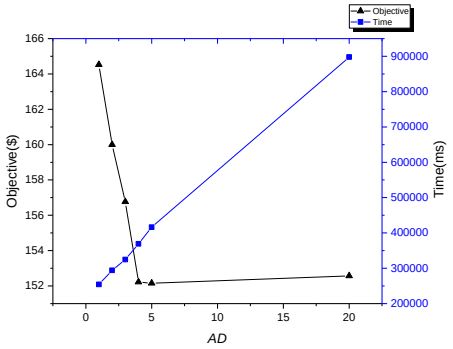

Fig. 6. Experimental results about parameter *AD*.

TABLE V
PARAMETER *AD* SELECTION BETWEEN 4 AND 5

AD	Mean	B/W	Mean	B/W
	M_1000_1		C_1000_1	
4	152.24		135.16	
5	152.08	E	137.42	E
	L_1000_1		S_1000_1	
4	825.57		560.22	
5	816.93	W	563.68	E

'B/W' represents that the results of *AD*=4 are significantly better/worse than *AD*=5 according a Wilcoxon rank sum test at level 0.05; 'E' represents that they get equal performance.

B. Investigation of the Parameter AD

AD is an important parameter during the solution construction. If it is set to a small value, the search space will grow in a low speed, so that the execution time of A-ACO can be relatively short. But the performance will decrease, since the exploration ability of the algorithm is limited. Contrarily, if it is set to a big value, the search space will grow rapidly, the execution time of A-ACO will be long, and the performance will increase. Here we use an xlarge instance M_1000_1 to find how this parameter affects the execution time and the performance. Since the deadline constraint is loose, basically A-ACO can always get feasible solutions so that we can compare the objective value.

AD is set to six values {1, 2, 3, 4, 5, 20}. Other settings are kept unchanged. Each configuration is tested 20 times to get the mean value. Experimental results are shown in Fig. 6.

From the figure, we can see that the execution time increases linearly with the growth of *AD*. Regarding the objective value, we can find that it decreases rapidly at the beginning. However, when *AD* increases to 5, the objective value stops decreasing and holds on that level thereafter. Converted the measurement from millisecond into minute, the execution time of A-ACO with *AD*=5 is 6.935 minutes which is still within an acceptable range. However only judging by M_1000_1, *AD*=4 also seems to be a rational choice since the difference between 4 and 5 is negligible on Fig. 6. To select an appropriate value between 4

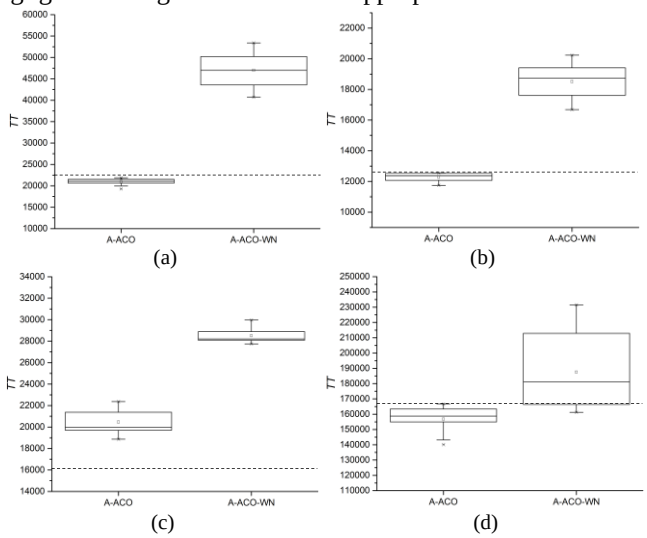

Fig. 7. Experimental results about the *nta* heuristic. (a) M_1000_3, (b) C_1000_3, (c) L_1000_3, and (d) S_1000_3.

TABLE VI
COMPARISON ON TC BETWEEN A-ACO AND A-ACO-WN

instance	M_100_1		C_100_1	
measure	median	Wilcoxon	median	Wilcoxon
A-ACO	15.0675	1.6E-1	14.875	4.96E-3
A-ACO-WN	15.645		14.4025	
instance	L_100_1		S_100_1	
measure	median	Wilcoxon	median	Wilcoxon
A-ACO	54.88	4.499E-2	50.995	5.9656E-1
A-ACO-WN	53.0425		50.5925	

and 5, C_1000_1, L_1000_1, and S_1000_1 are also tested. Each workflow is tested 20 times. Experimental results and explanations are shown in Table V. The results show that for M_1000_1, C_1000_1, and S_1000_1, setting *AD* to 4 or 5 does not have significant difference. However, for L_1000_1, setting *AD* to 5 is clearly better than 4. Thus, for the sake of universality, *AD* is set to 5 in the following experiments.

C. Investigation of the nta Heuristic

The dynamic heuristic *nta* along with the governor α are designed to help A-ACO meeting the deadline constraint. Here we are going to testify two questions, 1) whether this heuristic is effective in meeting the deadline constraint when the constraint is tight 2) and whether it will cause bad influence to the objective value (total cost *TC*) when the deadline constraint is easy to satisfy. An A-ACO without the *nta* heuristic is designed as the control group, denoted as A-ACO-WN.

To answer the first question, the four xlarge workflows with the most tight deadline constraint, M_1000_3, C_1000_3, L_1000_3, and S_1000_3 are used as the test cases. Results about the total execution time *TT* of these workflows are shown in Fig. 7 in the form of box chart. The dotted lines in the figures represent each instance's deadline. Observing the Fig. 7, we can find that basically the total time *TT* values achieved by A-ACO are smaller than A-ACO-WN got. On the first two instances, M_1000_3 and C_1000_3, A-ACO got feasible solutions in all 20 times run and A-ACO-WN never got a feasible solution. On L_1000_3, although both methods did not find feasible solutions, Fig. 7(c) shows that the range of A-ACO is still lower than the range of A-ACO-WN. The results show that the constraint is too tight for L_1000_3. Although the *nta* heuristic can help the algorithm meet the deadline, the main functionality of A-ACO is still to cut off the total cost rather than time. Thus when the deadline is too tight, A-ACO may still fail in finding feasible solutions. For such case, HEFT is recommended. On the last instance, A-ACO kept its good performance that all solutions found were feasible. On the contrary, only in a few runs, A-ACO-WN has found feasible solutions. The consequence is that the *nta* heuristic is truly useful in meeting tight deadline constraint.

To answer the second question, the four large workflows with the loosest deadline constraint, M_100_1, C_100_1, L_100_1, and S_100_1 are used as the test cases on which both A-ACO and A-ACO-WN can get feasible solutions. Results of the total cost TC are shown in Table VI. Wilcoxon rank sum test is made to show whether the results achieved by A-ACO and A-ACO-WN have significant difference. If we set the level to 0.05, according to Table VI, on M_100_1 and S_100_1,

TABLE VII
COMPARISON OF THE SUCCESS RATE AND TOTAL COST AMONG A-ACO, PSO, ACS, AND HEFT ON 48 TEST CASES

	Suc	B/W	Suc	B/W	Suc	B/W	Suc	B/W	Suc	B/W	Suc	B/W	Suc	B/W	Suc	B/W
workflow	M_25_1		M_50_1		M_100_1		M_1000_1		C_30_1		C_50_1		C_100_1		C_1000_1	
A-ACO	20		20		20		20		20		20		20		20	
PSO	20	W	20	B	20	B	20	B	20	E	20	B	20	B	20	B
ACS	20	E	20	B	20	E	20	W	20	E	20	E	20	E	20	B
BASE(HEFT)	20	B	20	B	20	B	20	B	20	B	20	B	20	B	20	B
workflow	M_25_2		M_50_2		M_100_2		M_1000_2		C_30_2		C_50_2		C_100_2		C_1000_2	
A-ACO	20		20		20		20		20		20		20		20	
PSO	20	E	20	B	20	B	20	B	20	E	20	B	20	B	20	B
ACS	20	E	4	NA	0	NA	1	NA	20	E	20	E	20	E	20	B
BASE(HEFT)	20	B	20	B	20	B	20	B	20	B	20	B	20	B	20	B
workflow	M_25_3		M_50_3		M_100_3		M_1000_3		C_30_3		C_50_3		C_100_3		C_1000_3	
A-ACO	20		20		20		20		20		20		20		20	
PSO	20	B	20	B	20	B	20	B	20	B	20	B	20	B	12	NA
ACS	20	E	3	NA	0	NA	0	NA	20	E	20	E	20	E	0	NA
BASE(HEFT)	7	NA	19	NA	20	B	20	B	20	B	20	B	20	B	20	E
workflow	L_30_1		L_50_1		L_100_1		L_1000_1		S_30_1		S_60_1		S_100_1		S_1000_1	
A-ACO	20		20		20		20		20		20		20		20	
PSO	20	W	20	B	20	B	0	NA	20	B	20	B	20	B	20	B
ACS	20	W	20	E	20	B	9	NA	20	E	20	E	20	E	20	B
BASE(HEFT)	20	B	20	B	20	B	20	B	20	B	20	B	20	B	20	B
workflow	L_30_2		L_50_2		L_100_2		L_1000_2		S_30_2		S_60_2		S_100_2		S_1000_2	
A-ACO	20		20		20		14		20		20		20		20	
PSO	20	W	20	W	19	NA	0	NA	20	B	20	B	20	B	20	B
ACS	20	W	20	W	19	NA	0	NA	20	E	20	E	20	B	20	B
BASE(HEFT)	20	B	20	B	20	B	20	NA	20	B	20	B	20	B	20	B
workflow	L_30_3		L_50_3		L_100_3		L_1000_3		S_30_3		S_60_3		S_100_3		S_1000_3	
A-ACO	20		20		20		0		20		20		20		20	
PSO	20	W	20	W	12	NA	0	NA	20	B	20	B	20	B	20	B
ACS	20	W	20	W	15	NA	0	NA	20	E	20	E	20	E	20	B
BASE(HEFT)	20	B	20	B	20	E	20	NA	13	NA	16	NA	20	B	20	B

"NA" means "not available". 'B' represents that the results of A-ACO are significantly better according to a Wilcoxon rank sum test at level 0.05; 'W' represents that the results of A-ACO are significantly worse according to a Wilcoxon rank sum test at level 0.05; 'E' represents that A-ACO got equal performance with the compared algorithm according to a Wilcoxon rank sum test at level 0.05. 'Suc' shows how many successful runs they got among total 20 runs.

A-ACO has similar performance with A-ACO-WN. On C-100-1 and L-100-1, A-ACO is slightly worse than A-ACO-WN. But the median values show that the gap between them are small. The consequence tells us that when the deadline constraint is relatively loose, the *nta* heuristic indeed has a little bit bad influence to the objective on some workflows. But considering its usage in handling the tight deadline constraint, we think that its deeds outweigh its faults.

D. Comparison with Other Methods

The overall results of all four methods are shown in Table SI in the supplemental material, including the number of runs in which they got feasible solutions, the best objective value, the mean value, whether A-ACO is better or worse than others according to the Wilcoxon rank sum test. For the sake of brevity and clarity, summerized information is shown in Table VII, including the success rate and the results of Wilcoxon rank sum test.

1) Comparison of the success rate

At first, we check the amount of the successful run of the algorithms on each workflow to get a view of their abilities to meet the deadline constraint. Observing Table VII, we can obtain the following findings:

a) A-ACO gets 100% success rate on 46 workflows; PSO gets 100% success rate on 42 workflows; ACS gets 100% success rate on 36 workflows.

b) For the workflows with the most relaxed deadline constraint, all four algorithms perform well, except for one case L_1000_1 on which PSO and ACS do not find feasible solutions in some runs.

c) Tightening the deadline constraint, we can find that the performances of ACS and PSO start decreasing on some workflows, such as M_1000_3, C_1000_3, L_1000_2, and L_1000_3. Compared with these two approaches, A-ACO is a little bit better where its success rate only decreases on the xlarge LIGO workflows. HEFT fails on some small instances like M_25_3 and S_30_3 rather than large workflows since in a small workflow, the critical path is more important to be scheduled right but the paralleled tasks. Due to the static heuristic, HEFT will not use expensive VMs for the tasks in critical path.

d) Overall, except on L_1000_2 and L_1000_3 where A-ACO does not get a 100% success rate, on the rest instances, A-ACO succeeds in every run. The consequence shows that the A-ACO's capability to satisfying the deadline constraint is better than ACS and PSO. Meanwhile it can also adapt to workflows with different scales.

2) Comparison of the total cost TC

Due to the restriction of the deadline constraint, it is meaningless to calculate the objective cost of the infeasible solution. Thus, only feasible solutions are concerned when dealing with the results about total cost. In the experiments, the best objective value (best) and the mean value (mean) are used as the standard of comparisons. Additionally, Wilcoxon rank sum test is conducted between A-ACO and the compared algorithms on the workflows where A-ACO makes 100% success rate and the success rate of HEFT or PSO or ACS is also 100%.

TABLE VIII
QUANTIFICATION OF TABLE VII

	PSO	ACS	BASE
Better	33	8	40
Worse	6	6	0
Equal	3	21	2
NA	6	13	6

First, we get a general view of the results based on the Wilcoxon rank sum test. The numbers about how many 'B', 'W', or 'E' A-ACO gets compared with the other three algorithms are shown in Table VIII. It is clear that PSO and HEFT are overwhelmed by A-ACO. ACS seems have similar performance with A-ACO, but on many workflows, it just fails in finding feasible solutions. Specific values about the objective (total cost) can be found in the supplemental material in which the numerical difference among the tested algorithms is shown.

Second, we analyze the results from the perspective of the workflow type, and we focus on the results of the large and xlarge workflows because their sizes are close to real applications. For Montage workflows, A-ACO performs significantly better on M_100_2, M_100_3, M_1000_2, and M_1000_3. On M_100_1 and M_1000_1, ACS performs better. Clearly, ACS degrades rapidly along with the growth of the deadline constraint on Montage workflows. For CyberShake workflows, A-ACO and ACS are well matched on the large workflows and totally dominate the other two algorithms. However, when the scale grows to xlarge, A-ACO becomes better than ACS. Additionally, we can see that the total cost obtained by A-ACO grows with the deadline constraint. Finally, on C_1000_3, it gets comparable results with HEFT which implies that the *nta* heuristic make A-ACO lease a lot of new VMs for tasks. When it goes to the LIGO workflows, we can see that the three metaheuristic algorithms get into trouble. With the increase of both scale and deadline constraint, PSO crashes at first, then ACS loses its functionality, finally on L_1000_3, A-ACO also fails to find a feasible solution. But an interesting thing is that HEFT performs very well on LIGO workflows. Checking the structure of LIGO, we find that there is a kind of task which occupy 98% of the execution time of the whole workflow, and most of them are in the same layer which means they can be executed in parallel. Thus the way to give them each a new VM in HEFT is effective in meeting the deadline constraint. Finally, SIPHT workflows show clear preference to the A-ACO approach. All approaches have achieved 100% success rate, and the Wilcoxon rank sum test shows that A-ACO has made wonderful job on xlarge SIPHT workflows. On S_100_1 and S_100_3, A-ACO and ACS tie for first place. On S_100_2, A-ACO still dominates the other three methods.

Overall, by checking the success rate and the total cost, we can make the conclusion that A-ACO is both able to meet different deadlines and effective to reduce the expenditure of executing cloud workflows.

E. Algorithm Complexity Analysis

In this subsection, we make analysis about the time

complexity of the four approaches. HEFT is a heuristic approach whose execution time is much shorter than metaheuristic algorithms. ACS, PSO, and A-ACO belong to the iterative approach which really takes time to find a near-optimal solution.

HEFT only generates one solution. If the time complexity of assigning one task is O(1), the time complexity of HEFT is O(n) where n the scale of the problem. In the problem studied in this paper, n is the task number of workflows. The PSO method used in [13] derives from the original PSO algorithm which is used to solve the continuous space optimization problem. For each dimension (task), the value (VM) is determined just by constant times of addition and multiplication. Thus for a single particle in one iteration, the time complexity to generate a solution is O(n). Assuming the swarm size is m, the maximum iteration number is k, then the time complexity of PSO is O($n \cdot m \cdot k$). The ACS method proposed in [29] derives from the ACS algorithm which is proposed to solve the combinatorial optimization algorithms. Its time complexity of generating a single solution depends on the value range of each dimension. Thus it considers every resource for each task's assignment. As shown by (5), the size of the resource pool is equal to the product of the maximum number of tasks that can run in parallel p and the number of VM types q. Thus for a single ant in one iteration, the time complexity to generate a solution is O($n \cdot p \cdot q$). The whole complexity of ACS is O($n \cdot p \cdot q \cdot m \cdot k$).

However, A-ACO modified the structure of the pheromone matrix and the solution construction method to reduce the value

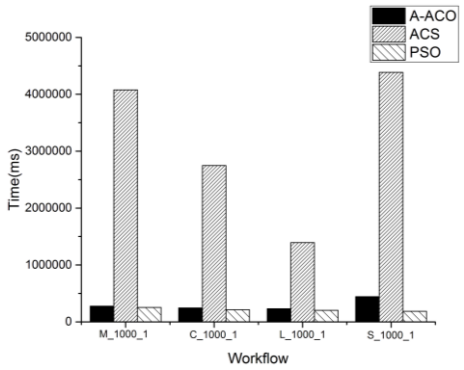

Fig. 8. Comparison of the real run time on four workflows. The x-axis represents the workflow name. The y-axis represent the execution time, measured in milliseconds.

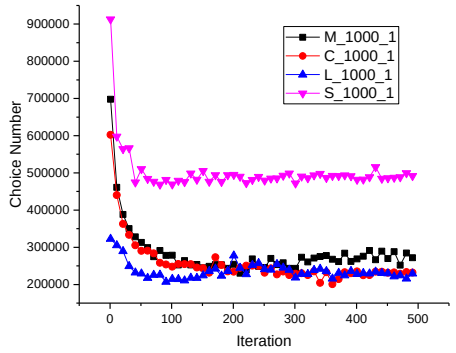

Fig. 9. Choice number to generate a solution on the four test cases of A-ACO. The x-axis represents the iteration number. The y-axis represents the choice number.

range of each dimension. More specifically, for the workflow scheduling problem studied in this paper, the time complexity of A-ACO is designed to depend on the size of the option set *os* which is mentioned in Algorithm 2. The growing speed of *os* size depends on the parameter *AD*. In the experiment we set *AD* to 5. Usually for a large workflow, the maximum number of tasks that can run in parallel p is about half of its task number, $p \approx n/2$, which means for a workflow with 1000 tasks, p is approximately equal to 500. Considering 20 different VM configurations, we size of the resource pool will be 20×500=10000. If the *os* size increases 5 every time (this is the worst situation, generally the size will not always increase 5 after assigning a task), it can only grows to 5000 which is half of the pool size. Thus, the time complexity of A-ACO is much smaller than O($n \cdot p \cdot q \cdot m \cdot k$/4).

Theoretically speaking, the time complexity of A-ACO is lower than ACS and is higher than PSO if m and k are considered identical. However, when we apply a meta-heuristic algorithm, besides the execution time of the algorithm itself, most time expenditure actually comes from the fitness evaluation, i.e. scheduling process. Following, the real run time of the four methods on four xlarge workflows, M_1000_1, C_1000_1, L_1000_1, and S_1000_1, using the computer with Core i3-3240 3.40GHz processor are collected. However, the execution time of HEFT is shorter than 100ms, which is too smaller to be displayed. Thus only the result of the three meta-heuristic methods are shown in Fig. 8. Comparing A-ACO with PSO, we can find that except S_1000_1, on the other three workflows, these two method take roughly the same time. The PSO algorithm uses more fitness evaluations, that is why the complexity of A-ACO is higher than PSO but they get similar execution time. To find the reason why A-ACO gets longer execution time on S_1000_1, we have counted how many choices are considered to make a solution in each generation. One 'choice' represents one available VM. The results are show in Fig. 9. It is clear that on S_1000_1, A-ACO faces more VM choices when constructing a solution, thus the execution will be long. This phenomenon originates from the characters of the tasks of S_1000_1. Their lower memory bounds are relatively smaller than the other test cases which does not exclude many VMs. Compared with ACS which utilizes same number of times of fitness evaluation, due to the novel solution construction method, the time consumption of A-ACO is less than a tenth of the time consumption of ACS. Thus, synthesizing all the experimental results, we can make the conclusion that the A-ACO approach is effective and efficient.

VI. Conclusion

In this paper, an intelligent cloud workflow scheduling system is proposed from the users' perspective to reduce the expenditure of utilizing IaaS cloud service. The main contributions are characterized in two aspects. First, models of the applications and computing resources are improved. The impact of main memory is taken into consideration which leads to new estimation method of execution time. Such

modifications make the model more practical so that the scheduling can be more accurate. Second, a new adaptive ACO variant named A-ACO is proposed in the scheduling module and two useful heuristic factors are employed. An auxiliary data structure, option set *os*, is used to decrease the search space of the problem and the complexity of A-ACO. Experimental results show that the method yields better results in both success rate and total cost. Run time analysis also shows that the proposed method is efficient.

In future research, more accurate computing models to characterize the workflow applications and computing resources are needed to improve the estimation module, which will be helpful in estimating the execution time of the tasks, thus facilitating the scheduling. Meanwhile, more efficient scheduling approaches are still required in the scheduling module to continue decreasing the execution time of the optimization algorithms, since the time consumption is still high on the large workflows. Also it will be useful to develop online systems to solve workflow scheduling problems from the perspective of suppliers, which will bring more convenience to promote the utility of the public IaaS cloud.

REFERENCES

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

[2] 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, 2009.

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

[4] A. Basu and A. Kumar, "Research commentary: Workflow management issues in e-business," *Inf. Syst. Res.*, vol. 13, no. 1, pp. 1-14, 2002.

[5] G.B. Berriman, E. Deelman, *et al*., "Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand," *Astronomical Telescopes and Instrumentation*, International Society for Optics and Photonics, pp. 221-232, 2004.

[6] I. Bertram, D. Evans, G.E. Graham, P. Love, R. Walker, "McRunjob: A High Energy Physics Workflow Planner for Grid Production Processing," in *Proc. UK e-Science All Hands Meeting*, 2003.

[7] I.J. Taylor, E. Deelman, D.B. Gannon, and M. Shields, *Workflows for e-Science: scientific workflows for grids.* Springer Publishing Company, Incorporated, 2014.

[8] B. Ludäscher, I. Altintas, C. Berkley, *et al*., "Scientific workflow management and the Kepler system," *Concurr. Comput. Pract. Exp.*, vol. 18, no. 10, pp. 1039-1065, 2006.

[9] M. Armbrust, A. Fox, *et al*., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 2010.

[10] W. Wang, Y. Jiang, and W. Wu, "Multiagent-Based Resource Allocation for Energy Minimization in Cloud Computing Systems," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 47, no. 2, pp. 205-220, 2017.

[11] P. Mell, T. Grance, "The NIST definition of cloud computing," *Special Publication 800-145*, NIST, Gaithersburg, 2001.

[12] E. Sousa, F. Lins, E. Tavares, P. Cunha, and P. Maciel, "A modeling approach for cloud infrastructure planning considering dependability and cost requirements," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 45, no. 4, pp. 549-558, 2015.

[13] M.A. Rodriguez, R. Buyya, "Deadline based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222-235, 2014.

[14] S. Ostermann, A. Losup, *et al.*, "A performance analysis of EC2 cloud computing services for scientific computing," *Cloud Comput.*, pp. 115-131, 2009.

[15] 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 CEC*, 2015, pp. 708-714.

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

[17] 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'l Conf. Adv. Inform. Netw. Appl.*, 2010, pp. 400-407.

[18] M. Rahman, R. Hassan, R. Ranjan, R. Buyya, "Adaptive workflow scheduling for dynamic grid and cloud computing environment," *Concurr. Comput. Pract. Exp.*, vol. 25, no. 13, pp. 1816-1842, 2013

[19] S.J. Xue, W. Wu, "Scheduling workflow in cloud computing based on hybrid particle swarm algorithm," *TELKOMNIKA Indonesian J. Electrial Eng.*, vol. 10, no. 7, pp. 1560-1566, 2012.

[20] C. Lin, S. Lu, "Scheduling scientific workflows elastically for cloud computing," in *Proc. IEEE int'l Conf. Cloud Comput.*, 2011, pp. 746-747.

[21] Z. Wu, Z. Ni, L. Gu, and X. Liu, "A revised discrete particle swarm optimization for cloud workflow scheduling," in *Proc. int'l Conf. Comput. Intell. Security*, 2010, pp. 184-188.

[22] L.F. Bittencourt, E.R.M. Madeira, "HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds," *J. Internet Services and Appl.*, vol. 2, no. 3, pp. 207-227, 2011.

[23] G.E. Suh, L. Rudolph, and S. Devadas, "Effects of memory performance on parallel job scheduling," *Job Scheduling Strategies for Parallel Process.*, 2001, pp. 116-132.

[24] E.S. Jung and R. Kettimuthu, "Challenges and Opportunities for Data-intensive Computing in the Cloud," *Computer*, vol. 47. no. 12, pp. 82-85, 2014.

[25] C.P. Chen, C. Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data," *Inform. Sciences*, vol, 275, pp. 314-347, 2014.

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

[27] Y.J. Gong, J.J. Li, Y.C Zhou, Y. Li, H.S.H. Chung, Y.H. Shi, and J. Zhang, "Genetic Learning Particle Swarm Optimization," *IEEE Trans. Cybern.*, vol. 46, no. 10, pp. 2277-2290, 2016.

[28] M. Dorigo, L.M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53-66, 1997.

[29] Z.G. Chen, Z.H. Zhan, H.H. Li, K.J. Du, J.H. Zhong, Y.W. Foo, Y. Li, J. Zhang, "Deadline Constrained Cloud Computing Resources Scheduling Through An Ant Colony System Approach," in *Proc. IEEE ICCCRI*, 2015, pp. 112-119.

[30] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28-39, 2006.

[31] D. Merkle, M. Middendorf, H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 333-346, 2002.

[32] L.N. Xing, P. Rohlfshagen, Y.W. Chen, and X. Yao, "A hybrid ant colony optimization algorithm for the extended capacitated arc routing problem," *IEEE Trans Syst. Man Cybern. B Cybern.*, vol. 41, no. 4, pp. 1110-1123, 2011.

[33] W.N. Chen and J. Zhang, "Ant Colony Optimization for Software Project Scheduling and Staffing with an Event-Based Scheduler," *IEEE Trans. Softw. Eng.*, vol. 39, no. 1, pp. 1-17, 2013.

[34] K.M. Sim and W.H. Sun, "Ant colony optimization for routing and load-balancing: survey and new directions," *IEEE Trans. Syst. Man Cybern. A Syst. Humans*, vol. 33, no. 5, pp. 560-572, 2003.

[35] Z. Cai, Y. Wang, "A multiobjective optimization-based evolutionary algorithm for constrained optimization," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 658-675, 2006.

[36] H. Topcuoglu, S. Hariri, and M.Y. Wu, "Task scheduling algorithms for heterogeneous processors," in *Proc. IEEE 8th Heterogeneous Computing Workshop*, 1999, pp. 3-14.

[37] M. Maheswaran, S. Ali, H.J. Siegal, D. Hensgen, and R.F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Proc. IEEE 8th Heterogeneous Computing Workshop*, 1999, pp. 30-44.

[38] C.K. Chang, H. Jiang, Y. Di, D. Zhu, and Y. Ge, "Time-Line Based Model for Software Project Scheduling with Genetic Algorithms," *Inf. and Softw. Technol.*, vol. 50, pp. 1142-1154, 2008.

[39] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," *Future Gener. Comput. Syst.*, vol. 48, pp. 1-18, 2015.
[40] A. Silberschatz, P.B. Galvin, and G. Gagne, *Operating system concepts*. Wiley, 2013.
[41] 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, no. 1/2, pp. 460-471, 2010.
[42] E. Walker, "Benchmarking Amazon EC2 for high-performance scientific computing," *USENIX login*, vol. 33, no. 5, pp. 18-23, 2008.
[43] A. Batat and D.G. Feitelson, "Gang Scheduling with Memory Considerations," in *Proc. 14th IEEE int'l Parallel. and Distrib. Process. Symposium*, 2000, pp. 109-114.
[44] M. Iverson, F. Özgüner, and G.J. Follen, "Run-time statistical estimation of task execution time for heterogeneous distributed computing," in *Proc. 5th IEEE int'l High Perform. Distrib. Comput. Symposium*, 1996, pp. 263-270.
[45] S. Krishnaswamy and S.W. Loke, "Estimating computation times of data-intensive applications," *IEEE Distrib. Syst. Online*, vol. 5, no. 4, 2004.
[46] M.K. Qureshi, V. Srinivasan, and J.A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24-33, 2009.
[47] Amazon EC2 Instance Types. Available: https://aws.amazon.com/ec2/instance-types/?nc1=h_ls
[48] K. Deb, "An efficient constraint handling method for genetic algorithms," *Comput. Methods Appl. Mech. Eng.*, vol. 186, nos. 2-4, pp. 311-338, 2000.
[49] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic," in *Proc. CEC*, 1999, pp. 1470-1477.
[50] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B Cybern.*, vol. 26, no. 1, pp. 29–41, 1996.
[51] T. Stützle and H.H. Hoos, "MAX–MIN ant system," *Future Gener. Comput. Syst.*, vol. 16, no. 8, pp. 889-914, 2000.
[52] G. Juve, A. Chervenak, *et al*., "Characterizing and profiling scientific workflows," *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682-692, 2013.
[53] R.N. Calheiros, R. Ranjan, *et al*., "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.

Ya-Hui Jia (S'14) received his Bachelor's degree from Sun Yat-sen University, China, in 2013, where he is currently pursuing his Ph. D. degree. He is also a research assistant with Scholl of Computer Science and Engineering, South China University of Technology, China. His current research interests include evolutionary computation algorithms and their applications on software engineering, cloud computing, and intelligent transportation.

Wei-Neng Chen (S'07-M'12) received the Bachelor's degree and the Ph.D. degree from Sun Yat-sen University, China, in 2006 and 2012, respectively. He is currently a professor with the School of Computer Science and Engineering, South China University of Technology, China. His current research interests include swarm intelligence algorithms and their applications on cloud computing, operations research and software engineering. Dr. Chen has published 50 papers in international journals and conferences. His doctoral thesis received the IEEE Computational Intelligence Society (CIS) Outstanding Dissertation Award in 2016. He also received Natural Science Foundation for Distinguished Young Scientists of Guangdong Province, China in 2015, the "Guangdong Special Support Program" for Outstanding Young Scientists in 2015, and the Pearl River New Star in Science and Technology in 2014.

Hua-Qiang Yuan received the Ph.D. degree from Shanghai Jiao Tong University, China in 1996. He is currently a professor with the School of Computer Science and Network Security, Dongguan University of Technology, China. His current research interests include computational intelligence, cyberspace security

Tianlong Gu received the M.Eng. degree from Xidian University, China, in 1987, and the Ph.D. degree from Zhejiang University, China, in 1996. From 1998 to 2002, he was a Research Fellow with the School of Electrical and Computer Engineering, Curtin University of Technology, Australia, and a Post-Doctoral Fellow with the School of Engineering, Murdoch University, Australia. He is currently a Professor with the School of Computer Science and Engineering, Guilin University of Electronic Technology, China. His research interests include formal methods, data and knowledge engineering, software engineering, and information security protocol.

Huaxiang Zhang is currently a professor with the School of Information Science and Engineering, Shandong Normal University, China. He received his Ph.D. from Shanghai Jiaotong University in 2004, and worked as an associated professor with the Department of Computer Science, Shandong Normal University from 2004 to 2005. He has authored over 100 journal and conference papers and has been granted 8 invention patents. His current research interests include machine learning, pattern recognition, evolutionary computation, web information processing, etc.

Ying Gao received the Bachelor's degree, Master's degree from Central South University of China and the Ph.D. degree from South China University of Technology, China, in 1997, 2000 and 2006, respectively. She is currently a professor with the School of Computer Science and Engineering, South China University of Technology, China. Her current research interests include Service-oriented computing technology, software architecture, and network security. Dr. Gao has published more than 30 papers in international journals and conferences.

Jun Zhang (M'02-SM'08-F'17) received the Ph.D. degree in Electrical Engineering from the City University of Hong Kong in 2002. From 2004 to 2016, he was a professor with SUN Yat-sen University. Since 2016, he has been with South China University of Technology, Guangzhou, China, where he is currently a Cheung Kong Chair Professor. He has authored seven research books and book chapters, and over 100 technical papers in his research areas. He is Fellow of Institute of Electrical and Electronics Engineers (IEEE). His current research interests include computational intelligence, cloud computing, big data, high performance computing, data mining, wireless sensor networks, operations research, and power electronic circuits.

Professor Zhang was a recipient of the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011 and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the IEEE Transactions on Evolutionary Computation, the IEEE Transactions on Industrial Electronics, and the IEEE Transactions on Cybernetics. He is the Founding and Current Chair of the IEEE Guangzhou Subsection and IEEE Beijing (Guangzhou) Section Computational Intelligence Society Chapters. He is the Founding and Current Chair of the ACM Guangzhou Chapter.