EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

An IoT-based task scheduling optimization scheme considering the deadline and cost-aware scientific workflow for cloud computing

Xiaojin Ma[1,2], Honghao Gao[3*], Huahu Xu[1,4] and Minjie Bian[1,4]

Abstract

Large-scale applications of Internet of things (IoT), which require considerable computing tasks and storage resources, are increasingly deployed in cloud environments. Compared with the traditional computing model, characteristics of the cloud such as pay-as-you-go, unlimited expansion, and dynamic acquisition represent different conveniences for these applications using the IoT architecture. One of the major challenges is to satisfy the quality of service requirements while assigning resources to tasks. In this paper, we propose a deadline and cost-aware scheduling algorithm that minimizes the execution cost of a workflow under deadline constraints in the infrastructure as a service (IaaS) model. Considering the virtual machine (VM) performance variation and acquisition delay, we first divide tasks into different levels according to the topological structure so that no dependency exists between tasks at the same level. Three strings are used to code the genes in the proposed algorithm to better reflect the heterogeneous and resilient characteristics of cloud environments. Then, HEFT is used to generate individuals with the minimum completion time and cost. Novel schemes are developed for crossover and mutation to increase the diversity of the solutions. Based on this process, a task scheduling method that considers cost and deadlines is proposed. Experiments on workflows that simulate the structured tasks of the IoT demonstrate that our algorithm achieves a high success rate and performs well compared to state-of-the-art algorithms.

Keywords: IoT, Application scheduling, Virtualization, Multiple constraints

1 Introduction

With the rapid development of science and technology, the requirement of large-scale computing cannot be separated from scientific applications or life services. Due to the geometric growth of information and the complexity of data processing, researchers in most disciplines face more challenges and opportunities than ever [1]. Many science applications, such as the Internet of things (IoT), gene sequencing, and earthquake prediction, are becoming increasingly dependent on high-performance computing and distributed storage. In particular, the IoT, which depends on a reliable computing environment, has been widely utilized in various fields [2]. As an innovative application mode, the IoT uses sensing technology, computing technology, and communication technology to connect massive devices to achieve interaction with the real world. However, in practical applications, the IoT obtains various information through perception and realizes intelligent management of different objects by sharing and exchanging data in real time. Due to its own capacity limitations, such as CPU, memory, and battery capacity, IoT applications face constraints in completing tasks that require substantial computing resources and energy consumption at the device end [3]. At the same time, analysis and decision-making based on big data at the infrastructure level of the IoT are difficult.

Meanwhile, cloud computing, a new computing architecture, has been widely applied in recent years. Cloud computing is a model that aims to provide a flexible heterogeneous resource pool through the network, and

* Correspondence: gaohonghao@shu.edu.cn
[3]Computing Center, Shanghai University, Shanghai 200444, China
Full list of author information is available at the end of the article

users can rent different resources on demand. These configurable computing resources are maintained by cloud providers and can be rapidly provisioned and released [4]. Users procure and release computing resources, which are usually virtual machines (VMs) with different specifications, according to their own needs within a certain time. As two new technologies based on the Internet, the IoT and cloud computing are closely related in terms of their roles. The IoT intelligently manages equipment and digitalizes various information, and cloud computing can be used as a carrier for high-speed data processing, storage, and utilization. Cloud computing has provided the advantages of speed, convenience, and security that the IoT lacks, and the technology makes the real-time dynamic management and intelligent analysis of the IoT more reliable.

Based on the infrastructure as a service (IaaS) model of cloud computing, this paper studies the task scheduling of the IoT applications in the cloud environment. In this paper, the data and processing requirements of the IoT are expressed as different applications that are usually represented as workflows that consist of computing tasks and data edges generally organized as directed acyclic graphs (DAGs) [5]. Although implementing IoT applications in cloud computing has various advantages, some issues remain to be addressed. First, workflow scheduling is regarded as an NP-hard problem [6]. In particular, workflow scheduling refers to the assignment of tasks to virtual resources according to execution sequence [7]. Second, when a computing resource (VM) is leased or released, a proper handoff takes time. Since cost is an important indicator when applying workflows to cloud platforms [8], the lease time of the VM must be considered because the time will affect the cost and deadline of the workflow. Third, the performance of a VM can vary because of the virtualization and instability of hardware, multiuser environment, and other reasons. The overall CPU performance of a VM can be up to 24% in the Amazon public clouds, and the performance variation of a cloud can reach 30% in terms of makespan [9, 10]. The performance variation of VMs increases the execution makespan and cost of the workflow, leading to violation of the quality of service (QoS) in the cloud.

In this paper, based on the advantages and issues discussed above, we propose a deadline and cost-aware genetic algorithm (DCGA) for workflow scheduling in IoT applications. The presented solution improves the meta-heuristic genetic algorithm (GA) [11] by using new heuristic methods to schedule each task to execute on an appropriate VM. First, the workflow scheduling model in a cloud environment is constructed. Then, we describe the optimization goal and major conceptual definitions by formalizing definitions. Next, the proposed algorithm is described in detail in different stages, and the key links are illustrated with examples, such as encoding, crossover, and mutation. Finally, according to the characteristics of VMs in Amazon cloud services, the simulation platform is built to evaluate the performance of the proposed algorithm.

The main contributions of this paper are as follows:

- Most characteristics of cloud are considered, including on-demand acquisition, dynamic extension, heterogeneity, acquisition delay, performance variation of VMs, and pay-as-you-go.
- Novel schemes are designed for population initialization, selection, crossover, and mutation of the genetic algorithm, which improve the search performance in the solution space.
- A comparison of state-of-the-art algorithms and the proposed algorithm is performed on the simulation platform using different workflow specifications.

The remainder of this paper is organized as follows. Section 2 surveys and analyses related work. Section 3 presents the structure of the platform and the problem of scheduling. Then, the DCGA scheduling algorithm is proposed in Section 4, and empirical studies are presented in Section 5. Finally, Section 6 concludes the paper and discusses future work.

2 Related work

The workflow in IoT applications is usually represented as a DAG, in which the vertexes of the graph represent tasks and edges represent the relationship between two tasks and data transmission. Moreover, workflow scheduling can be separated into two stages: task selection and resource allocation. First, the task selection phase aims to determine the sequence of tasks according to the constraints in the workflow [12]. Tasks can be grouped in different ways as long as successive dependencies are not violated. Therefore, the task execution order can be arranged in many ways. Then, the resource allocation phase aims to choose the number and type of VMs to execute tasks [13]. In this phase, the number of VMs in the cloud is regarded as unlimited: a task can run on only one VM and a VM can run only one task at a time.

Workflow scheduling in distributed systems has been widely studied. Yu et al. [14] propose a cost-based workflow scheduling algorithm for utility grids. The algorithm minimizes the cost of execution while meeting a deadline. They use a Markov decision process approach to schedule task and reschedule unexecuted tasks to adapt to delays in service execution that may violate the deadline constraints. In another work, Yu et al. [15] use multi-objective evolutionary algorithms to obtain a set of feasible solutions after the user defines the deadline and budget constraints. The main objective is to let the user choose the best solution from multiple possible solutions. In [16], Sakellariou et al. implement an algorithm to schedule DAGs on heterogeneous machines under budget constraints. The algorithm computes the weight value of each task and machine via two approaches and then uses the weights to assign tasks to machines in consideration of the cost and budget. Using the concept of game theory and sequential cooperative game, Duan et al. provide two algorithms (game-quick and game-cost) to optimize performance and cost in [17]. In addition, they design and implement a novel system model with better controllability and predictability of multi-workflow optimization problems in a grid environment. Chen et al. proposed an ant colony optimization algorithm to schedule large-scale workflows with three QoS parameters in a grid computing scenario in [18]. The algorithm enables users to specify two of the constraints and finds an optimized solution for the third constraint while meeting these parameters. These various algorithms are designed for users in a distributed environment, and the cost is usually based on all services used.

With the rise and development of cloud computing, research has increasingly turned to this field. Unlike in grid and cluster computing, task scheduling in cloud computing not only needs to consider the makespan factor but also needs to reduce the cost, which is based on the lease intervals. Hoffa et al. [19] compare the performance of running the Montage workflow on various types of resources, including both physical and virtual environments. They note that virtual environments can provide the necessary scalability, while local environments are sufficient but are not a scalable solution. Byun et al. [20] propose partitioned balanced time scheduling for executing tasks on the minimum resources under a given deadline. The algorithm associates the workflow management system with the resource provisioning environment to minimize the execution cost of a workflow. Their approach takes advantage of the elasticity, but ignores the heterogeneity, of cloud resources. Balanced time scheduling (BTS) is proposed for estimating the minimum number of resources needed to execute tasks under a deadline in [21]. The BTS algorithm uses list scheduling technology to delay a task until reaching the latest execute time. According to the local optimal time of each task, the tasks are allocated to the same host as much as possible to achieve resource minimization in workflow scheduling. However, the researchers simply consider VMs of the same type in the cloud environment, and the network contention influencing the start time of the task is ignored. In [22], Malawski et al. address the scheduling problem and resource provisioning for workflows on IaaS. The algorithms aim to maximize the number of workflows executed under the QoS constraints, such as deadline and budget. However, the execution time of a task on a VM is regarded as a constant. Abrishami et al. [23] propose a static algorithm named IaaS cloud partial critical path (IC-PCP) to schedule tasks on IaaS. First, the algorithm calculates the latest finish time of each task and then assigns tasks on a partial critical path to the least expensive VM instance that can complete the tasks before deadlines. If the available VMs cannot satisfy the deadline constraints, the algorithm generates a new VM instance that meets the conditions to execute all the tasks. This process is repeated until all the tasks are scheduled. However, the algorithm uses task-level optimization instead of a global optimization and may fail to generate a better solution without considering the entire workflow structure and characteristics.

Due to the uncertainty in the number and scale of tasks and resources in cloud computing, meta-heuristic algorithms such as ant colony optimization (ACO), particle swarm optimization (PSO), and GA have been applied to scheduling problems. These methods obtain near optimal solutions via iteration. Although the complexity of these algorithms is more than that of heuristic and traditional algorithms, the methods have attracted substantial attention due to their good performance. Pandey et al. [24] try to achieve load balancing between VMs using a PSO-based algorithm that can minimize the execution cost of tasks. The researchers obtain the total cost of a single workflow by changing the communication and execution cost in the resource pool. However, the makespan of the workflow, which is ignored in the scheduling, may violate the deadline constraint when using the cost-minimization policy. Yao et al. [25] design an improved multi-objective PSO algorithm called endocrine-based coevolutionary multi-swarm for multi-objective optimization (ECMSMOO) to schedule the workflow in the cloud. An evolutionary strategy inspired by the endocrine regulation mechanism is designed for every particle to optimize different objectives, such as cost, makespan, and energy. Then, the algorithm uses competition and cooperation among swarms to avoid falling into a local optimum. However, the experiments are based on a limited set of VMs, which fails to take advantage of the infinite extensibility of the cloud. In [26], a PSO-based algorithm is established in the cloud computing. Based on the related theory of multi-core processors, the proposed method assigns tasks to processors via certain rules in a dynamic scheduling strategy. The algorithm focuses on the utilization and bandwidth of resources in the cloud environment without considering the cost and deadline, which are more important factors for users. Wu et al. [27] propose two algorithms named ProLis and L-ACO to minimize the execution cost under the deadline constraint of workflow scheduling on the cloud. The ProLis algorithm distributes a deadline to each task, ranks the tasks, and sequentially allocates each task to satisfy the QoS. Based on this, L-ACO employs ACO to construct different task order lists and to construct solutions that minimize the cost under the deadline. However, the start-up time and performance variation of VMs are not considered.

In recent years, increasing research has focussed on the characteristics of the cloud environment, such as indefinite quantity, heterogeneity, performance variation, and acquisition delay of VMs, and few of these characteristics have been fully considered in previous studies. Mao et al. [28] present a new auto-scaling mechanism named scaling-consolidation-scheduling (SCS) to allocate all workflow tasks to the most cost-efficient VMs. The researchers consider the characteristics of VMs, such as performance variation and acquisition delay, but ignore the data transfer time between tasks, which will affect the completion time and total execution cost of the workflow. Rodriguez et al. [29] propose a static cost-minimization and deadline-constrained algorithm for workflow scheduling in a cloud environment. Based on the main characteristics of IaaS, the researchers merge and model both resource provisioning and scheduling as an optimization problem and use PSO to generate a solution that minimizes the lease cost of VMs before the deadline. However, the index of resources used to encode particles does not include much information about the type of VMs, so the algorithm cannot easily produce the best global optimal solution when particles move to the individual best solution. Poola et al. [30] propose robustness-cost-time (RCT), robustness-time-cost (RTC), and weighted algorithms to schedule workflow tasks on heterogeneous resources in the cloud. These algorithms based on partial critical paths (PCPs) provide a robust and fault-tolerant schedule while minimizing the total makespan and cost. Moreover, the three policies have different objectives, robustness, time, and cost, and each objective has different priorities in each algorithm. However, these algorithms schedule the entire tasks of PCPs on VMs and may affect the makespan of the workflow because the performance variation of a VM will delay the completion time of all the tasks assigned to it, thereby affecting the start time of the tasks of the other PCPs. Sahni et al. [31] propose a dynamic cost-effective deadline-constrained heuristic algorithm called just in time (JIT-C) for the workflow scheduling problem. The algorithm aims to determine a feasible solution in which the resources are provisioned immediately before the tasks are ready to execute. Compared with the similar methods in [23, 30], this algorithm consumes more time for the pre-processing and monitoring control loop. To address the characteristics of task allocation in the cloud, Arabnejad et al. [32] introduce the budget deadline aware scheduling (BDAS) algorithm to perform workflow scheduling in IaaS clouds. A tunable cost-time tradeoff is proposed to satisfy both budget and deadline constraints. The researchers use deadline bottom level (DBL) and deadline top level (DTL) to allocate tasks to different levels, and the budget and deadline are proportionally distributed to each level. Then, the algorithm assigns each task to an instance in consideration of a tradeoff between makespan and cost. In the experiments, the authors adopt a 97-s boot time for resource allocation but ignore the performance variation of VMs, which is one of the most important factors affecting the deadline and cost.

According to these studies, the characteristics of task allocation in the cloud have become an indispensable concern in scheduling problems. Meanwhile, the meta-heuristic GA is often applied to the problem because of its time efficiency [33], and different improvements of the GA to solve the workflow scheduling problem have been studied in [34–37]. On the basis of these studies, this paper proposes a DCGA for workflow scheduling in the cloud while considering the characteristics of the cloud, such as on-demand acquisition, dynamic extension, heterogeneity, acquisition delay, and performance variation of VMs. The DCGA uses a novel method for population initialization, crossover, and mutation to minimize the total execution cost under a deadline constraint.

3 Problem definition

3.1 Application model

The workflow of an IoT application can be represented as a DAG $G = (T, E)$, where $T = \{t_1, t_2, t_3, \ldots, t_n\}$ is a set of vertices representing the tasks and $E = \{e_{ij} \mid t_i, t_j \in T\}$ is a set of edges denoting the control dependencies between tasks. Each task t_i represents an indivisible transaction with a certain computation workload (w_i). A dependency e_{ij} with transfer data (data_{ij}) is a precedence constraint between task t_i and task t_j. This constraint implies that the child task t_j can start only after the parent task t_i is finished and all associated data are transferred to t_j. Each task may have one or more parents (predecessors) or children (successors), and a child task cannot be executed unless all its parent tasks have been completed and all the transfer data have been received.

The workflow starts with the entry tasks and concludes with the exit tasks. Thus, we add two dummy tasks with zero execution and communication time to the DAG when there is more than one entry or exit task. The two tasks are represented as t_{entry} and t_{exit}, respectively. In addition, we define D as the deadline constraint to complete the execution of the workflow, and an elastic factor that will be introduced in the experimental section is used to adjust the limit time bounded by D. More details are provided in Section 4, and a sample workflow is illustrated in Fig. 1.

For the workflow shown in Fig. 1, different task allocation schemes will lead to different execution times and costs. For example, as shown in Fig. 2, this sample workflow is scheduled on three different VMs. In schedule plan 1, the data transfer time delays the start time of t_6 and t_7, which causes more idle time of V_2 and V_3. Thus, the total execution time and cost of the workflow are both increased. In contrast, the strategy of assigning tasks with large data transmission to the same VM can effectively reduce the waiting time of tasks in schedule plan 2. Thus, with improving the utilization of VMs, the total execution time and cost of the workflow have been greatly reduced.

It can be seen that the execution time and cost of workflow will greatly differ due to the different scheduling schemes. This situation will become more obvious and complex with an increase of the number of tasks in a heterogeneous cloud environment. Therefore, a feasible workflow scheduling solution will not only reduce the time and cost of the workflow but will also improve resource utilization in the cloud environment.

3.2 System model

We adopt the IaaS cloud as the system model to support workflow scheduling in this article. The IaaS service model provides various computing resources in the form of a set of VMs $\text{VM} = \{vm_1^2, vm_2^3, vm_3^1, \ldots vm_m^k\}$ used to execute the tasks of the workflow. The vm_m in the VM set can be classified as a VM type (vm^k) provided by the cloud provider. These VM types have different CPU types (processor_k) with different costs (cost_{vm^k}) per time interval (*time_interval*). Therefore, VM types with faster computing performance usually have higher costs. The pricing model is dependent on a pay-as-you-go billing scheme, and the users are charged for the number of time intervals within a VM leased period. Partial utilization is rounded to a full unit time, that is, when 1 h is the minimum time interval, the user has to pay for the entire hour even if a VM is used for only a few minutes. Moreover, data transfer is free in most cloud environments, so the corresponding cost is not considered in the model. Additionally, an unlimited number of VMs is assumed to be leased by a workflow. The initial boot time (*init_time*) is the VM acquisition delay when a VM is leased before it is available to execute a task. The performance variation (*per_vary*) of VMs, which is caused by virtualization of resources, the shared nature of the infrastructure, and other factors, must also be considered. The shutdown time when a VM is released is ignored because the time has a negligible impact on the workflow scheduling process. The average bandwidth between VMs in a cloud is always assumed to be a fixed value. Furthermore, the data transfer time between two tasks is calculated based on the quantity of data transferred, and when two tasks are assigned to the same VM, this time is considered to be zero.

3.3 Problem formulation

Based on the models discussed above, we define some notation in Table 1. The detailed descriptions are presented as follows.

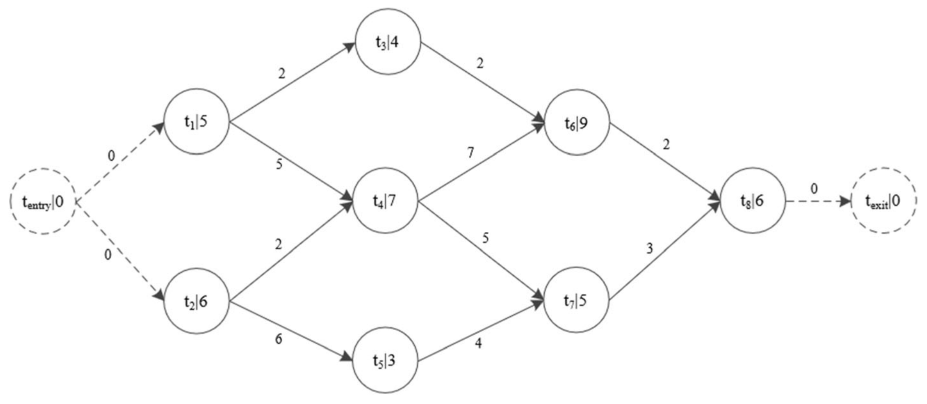


Fig. 1 A sample workflow model

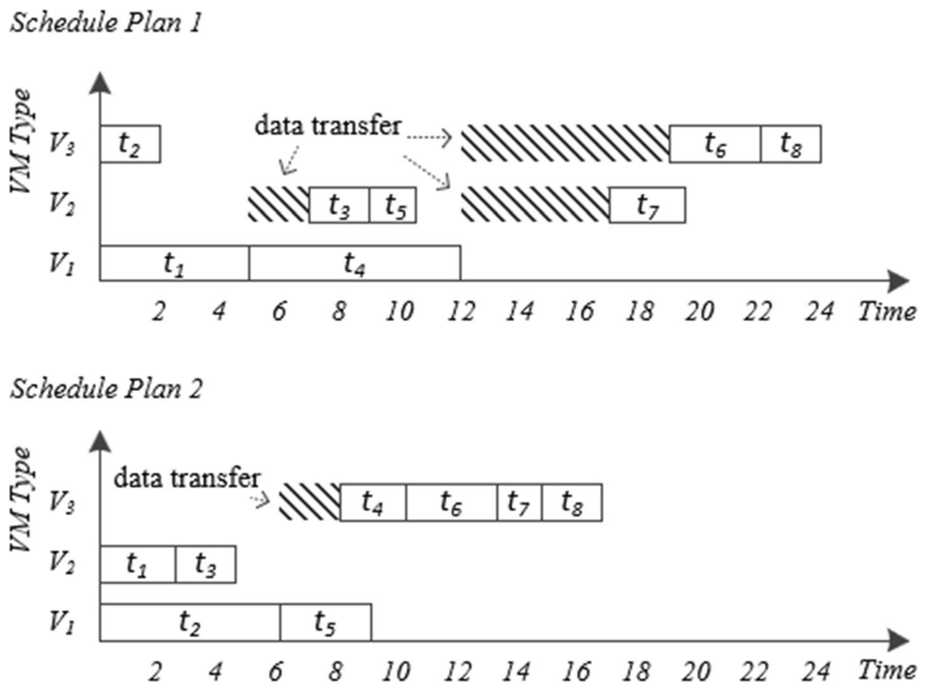


Fig. 2 Schedules of the workflow from Fig. 1 with different task assignment strategies

1. Predecessors pred(t_i) and successors succ(t_i): the immediate predecessors and successors of task t_i are defined as

$$\mathrm{pred}(t_i) = \{t_p | \exists e_{pi} \in E\} \tag{1}$$

$$\mathrm{succ}(t_i) = \{t_s | \exists e_{is} \in E\} \tag{2}$$

2. Estimated execution time $ET(t_i, vm^k)$ and data transfer time $TT(e_{ij})$: when task t_i is scheduled on a VM of type vm^k, the estimated execution time is defined as

$$ET\left(t_i, vm^k\right) = \frac{w_i}{\mathrm{processor}_k} \tag{3}$$

Before a task is executed, all the transfer data from its predecessors must be received. The data transfer time between task t_i and its parent task t_p is defined as

$$TT\left(e_{pi}\right) = \begin{cases} 0, \textit{if } t_p, t_i \text{ are scheduled on the same VM} \\ \dfrac{\mathrm{data}_{pi}}{\mathrm{bandwidth}}, \text{otherwise} \end{cases} \tag{4}$$

3. Latest finish time $LFT(t_i)$ and latest start time $LST(t_i)$: the latest finish time of task t_i is the time that t_i can complete its execution on the fastest VM type so that the makespan of workflow is no more than the deadline D. The definition of the latest start time is similar. The $LFT(t_i)$ and $LST(t_i)$ are obtained as

$$LFT(t_i) = \begin{cases} D, \text{if } t_i \text{ is the exit task } t_{\mathrm{exit}} \\ \min\limits_{t_s \in \mathrm{succ}(t_i)} \{LST(t_s) - TT(e_{is})\}, \text{otherwise} \end{cases} \tag{5}$$

$$LST(t_i) = LFT(t_i) - ET\left(t_i, vm^{\mathrm{fastest}}\right) \tag{6}$$

4. Start time $ST(t_i)$ and finish time $FT(t_i)$: the start time of task t_i on VM vm_m^k depends on the finish time of all predecessors of t_i, the data

Table 1 Notation

Symbol	Meaning
$G = (T, E)$	G is the directed acyclic graph representing a scientific workflow, $T = \{ t_1, t_2, t_3, \ldots, t_n \}$ is a set of tasks, and $E = \{ e_{ij} \mid t_i, t_j \in T \}$ is a set of edges, where n is the number of tasks in the workflow
D	The deadline of the workflow
w_i	The computation workload of task t_i
data_{ij}	The transfer data between task t_i and t_j
$VM = \{vm_1^2, vm_2^3, \ldots, vm_m^k\}$	VM is the set of VMs in the cloud, where m is the identifier of the VM and k is the VM type
$cost_{vm^k}$	The cost of VM type vm^k
$time\_interval$	Minimum time interval for billing when leasing a VM
$bandwidth$	The average bandwidth between VMs
$init\_time$	The acquisition delay of a VM
processor_k	The CPU processing performance of VM type vm^k
per_vary	Percentage of performance variation of a VM
$\text{pred}(t_i)$	List of all immediate predecessors of task t_i
$\text{succ}(t_i)$	List of all immediate successors of task t_i
$ET(t_i, vm^k)$	Estimated execution time of task t_i scheduled on a VM of type vm^k
$TT(e_{pi})$	Data transfer time between task t_p and t_i
$LST(t_i)$	Latest start time of task t_i
$LFT(t_i)$	Latest finish time of task t_i
$ST(t_i)$	Start time of task t_i on a VM
$FT(t_i)$	Finish time of task t_i on a VM
$LV(t_i)$	The level number l associates a task to a BoT
$BoT(l)$	A set of tasks that have the same level number l
$Avail(vm_m^k)$	The ready time of VM vm_m^k to execute a new task
$VLST(vm_m^k)$	Lease start time of VM vm_m^k
$VLFT(vm_m^k)$	Lease finish time of VM vm_m^k
TEC	Total execution cost of a workflow
TET	Total execution time of a workflow

transfer time between t_i and all its predecessors, and the available time of vm_m^k. The finish time is influenced by the percentage of performance variation per_vary of vm_m^k. The recurrence relations are

$$ST(t_i) = \begin{cases} 0, \text{if } t_i \text{ is the entry task } t_{\text{entry}} \\ \max\left\{\text{Avail}\left(vm_m^k\right), \max\limits_{t_p \in \text{pred}(t_i)} \left\{FT\left(t_p\right) + TT\left(e_{pi}\right)\right\}\right\}, \text{otherwise} \end{cases} \tag{7}$$

$$FT(t_i) = ST(t_i) + \frac{w_i}{(1-per\_vary)\text{processor}_k} \tag{8}$$

5. Available time $Avail(vm_m^k)$: the available time of VM vm_m^k is the ready time to execute a new task. Assume task t_i is the last scheduled task on vm_m^k. The $Avail(vm_m^k)$ will be updated to $FT(t_i)$. The definition is the same as in Eq. (8). If the VM is new and has no task assigned to it, its available time is equal to the initial boot time ($init\_time$).
6. VM lease start time $VLST(vm_m^k)$ and VM lease finish time $VLFT(vm_m^k)$: the lease start time of VM vm_m^k is the time when vm_m^k is ready to execute tasks, which is equal to $ST(t_i)$ when task t_i is the first executed task on vm_m^k. The VM lease finish time is the time at which vm_m^k is recovered by the cloud, which is equal to $FT(t_j)$ when task t_j is the last executed task on vm_m^k.
7. Total execution time TET and total execution cost TEC: the total execution time of the workflow is equal to the finish time of task t_{exit}, and the total execution cost is the total cost for all of the leased VMs.

Finally, the problem of workflow scheduling can be defined as finding a feasible solution for the given workflow such that the TEC is minimized and the TET is no greater than the deadline D. The problem is defined as

$$\text{Minimize } TEC(G) = \sum\nolimits_{m=1}^{M} \text{cost}_{vm^k} * \left\lceil \frac{VLFT\left(vm_m^k\right) - VLST\left(vm_m^k\right)}{time\_interval} \right\rceil$$
$$\text{Subject to } TET(G) = FT(t_{\text{exit}}) \le D \tag{9}$$

When deploying IoT applications to cloud computing, in addition to the cost of the VMs leased, the need to ensure that the results can be obtained within a specified time should also be considered. Therefore, under the restriction of deadline D in formula (9), the workflow can be guaranteed to meet the requirement of completion time while reducing the execution cost.

4 The proposed DCGA algorithm

In this section, based on the GA, the proposed deadline and cost-aware optimization approach to achieve the goal of Eq. (9) is described. The GA, a metaheuristic algorithm inspired by the ideas of natural selection and genetic evolution, is frequently applied to optimization problems. The basic idea of the GA is to initialize a certain number of populations whose individuals are called chromosomes that represent a solution. The chromosome genes can be encoded by bits 0 and 1. Individuals with better evaluation values are selected for crossover to form new chromosomes, and mutation is performed with a specified probability to vary the diversity of the next generation. In this way, after multiple iterations or upon reaching the termination condition, the individual with the maximum/minimum evaluation value is selected as the approximate optimal solution. Based on the problem definition in Section 3, the pseudocode of DCGA is shown in Algorithm 1, and the relevant operations are presented as follows. Table 2 defines the symbols used in these algorithms.

Compared to the traditional genetic algorithm, the proposed algorithm not only improves the coding but also reflects the topological structure between tasks, heterogeneity, and elasticity of the cloud environment through three strings of chromosomes. In the population initialization phase, the search scope can be reduced by adding specific individuals. In addition, the proposed algorithm improves the crossover and mutation operations to avoid the premature convergence problem of the traditional GA.

Algorithm 1. DCGA()

```
1.  TaskInitialization()
2.  currentPop←PopulationInitialization($T$, $V^K$)
3.  while(i<maxIter)
4.    while(the size of nextPop<popSize)
5.      parent chromosomes P1, P2←tournament selection in
        currentPop
6.      if random(0, 1)>crossRate
7.        add into nextPop←Crossover(P1, P2)
8.      else
9.        add P1, P2 into nextPop
10.     endif
11.     if random (0, 1)>mutationRate
12.       X←randomly choose a chromosome in nextPop
13.       add into nextPop←Mutation(X)
14.     endif
15.   endwhile
16.   currentBest←the best fitness value chromosome in nextPop
17.   if currentBest is better than globalbest
18.     globalbest←currentBest
19.   endif
20.   currentPop←nextPop
21.   i++
22. endwhile
```

Table 2 The parameters of DCGA

Symbol	Description
popSize	The size of the population
maxIter	The maximum number of iterations
i	The number of iterations
crossRate	The rate of crossover
mutateRate	The rate of mutation
initPop	The initial population
currentPop	The current population
nextPop	The next generation population
currentBest	The best chromosome in the current population
globalBest	The best chromosome in all iterations

4.1 Task initialization

Tasks in the DAG have certain topological structures and dependencies. To schedule tasks according to an order, tasks are separated into a bag of tasks (BoT) series according to their level l. That means each task in the same BoT has the same level, and there is no interdependence between them. The pseudocode is shown in Algorithm 2.

Algorithm 2. TaskInitialization()

```
1.  TempT←the temp set of ready tasks
2.  BoT(l)←a set of tasks with the same level l
3.  $LFT(t_{exit})$←the deadline D of workflow
4.  l=1
5.  add the exit task to BoT(l) and TempT
6.  while TempT is not null
7.    $t_i$←randomly remove a task from TempT
8.    for each task $t_p$ with a dependency e from $t_p$ to $t_i$
9.      remove edge e from the DAG
10.     if $t_p$ has no outing edge
11.       $l=LV(t_p)=LV(t_i)+1$
12.       add $t_p$ to BoT(l) and TempT
13.       calculate the latest finish time $LFT(t_p)$ by formula (5)(6)
14.     endif
15.   endfor
16. endwhile
```

We use depth-first search (DFS) [38] to generate topological sorts of tasks and use the DBL [39] to allocate tasks into different levels. The level of a task represents the maximum number of edges from it to the exit task. The level l can be defined as

$$LV(t_i) = \begin{cases} 1, \text{if } t_i \text{ is the exit task } t_{\text{exit}} \\ \max\limits_{t_s \in \text{succ}(t_i)} \{LV(t_s)\} + 1, \text{otherwise} \end{cases} \tag{10}$$

All tasks are then grouped into a set of $BoTs$ based on their levels

$$BoT(l) = \{t_i | LV(t_i) = l\} \tag{11}$$

where l is a level integer in [1, $LV(t_{\text{entry}})$].

4.2 Encoding

Workflow scheduling in a heterogeneous cloud environment can usually be divided into three stages: task execution order, task assignment to VM, and VM type matching. Then, a solution to the scheduling problem consists of three strings: the task order, the assigned VM, and the matched type. These strings represent the structure of the chromosomes in the GA. In this paper, we use an encoding method similar to that presented in previous literature [34].

The lengths of the three strings are set as the number of tasks. The first string is named *Task-Order*, and the value represents the order in which tasks are scheduled according to the topological structure of the DAG. Tasks in the string can switch positions as long as their level numbers l are equal. The next string, called *Task-VM*, denotes the VM to which the corresponding task is assigned. Similarly, the third string represents the type of the corresponding VM and is called *Type-VM*.

Omitting the entry and exit task, Fig. 3 shows the encoding of a possible schedule of the DAG in Fig. 1. In this case, the task scheduling sequence is {t_2, t_1, t_4, t_3, t_5, t_7, t_6, t_8} according to the *Task-Order* string. Each task in the sequential list is scheduled on the VM list { $vm_1^3, vm_5^4, vm_2^1, vm_3^3, vm_6^2, vm_4^1, vm_1^3, vm_2^1$ }, in which the type of each VM is presented in the *Type-VM* string. For example, task t_6 is assigned to vm_1 with type vm^3 after task t_5 is assigned to vm_6 with type vm^2. Meanwhile, tasks t_2 and t_6 are assigned to the same VM vm_1^3.

The method of representing the task order, assigned VM, and VM type by three strings not only reflects the topological structure of the workflow, heterogeneity, and elasticity of the cloud environment, but also can generate different scheduling schemes by adjusting each string. This coding method is helpful to improve genetic operations such as crossover and mutation to improve the efficiency of finding the optimal solutions.

4.3 Population initialization

The scale of the workflow directly affects the solution space of the scheduling problem. To reduce the search scope and accelerate the convergence of the solution, we propose a novel method of population initialization to produce task sequences and VM allocations that is different from those in [34–36]. In the proposed DCGA, the following steps are used to generate an effective initial population and complete the chromosome coding. The pseudocode is shown in Algorithm 3.

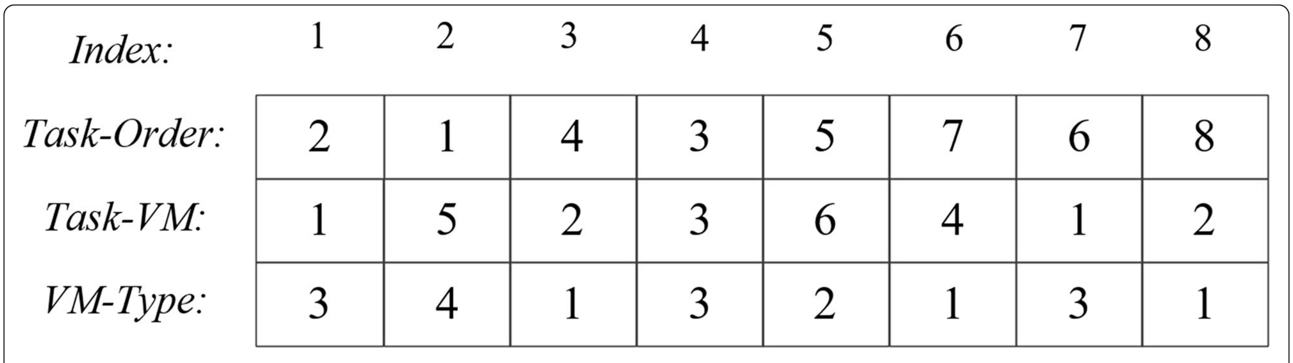
Algorithm 3. PopulationInitialization()

```
1.  initPop←null
2.  while the size of initPop<popSize
3.      for $l$ from $LV(t_{entry})$ to 1
4.          while $BoT(l)$ is not null
5.              $t_i$←randomly remove a task from $BoT(l)$
6.              add $t_i$ into Task-Order
7.          endwhile
8.      endfor
9.      Task-VMA←a string of different numbers
10.     VM-TypeA←a string of the fastest VM type
11.     minMakespanChrom←a chromosome with Task-Order; Task-VMA
        and VM-TypeA
12.     Task-VMB←a string of zeros
13.     VM-TypeB←a string of the cheapest VM type
14.     minCostChrom←a chromosome with Task-Order; Task-VMB and
        VM-TypeB
15.     for each $t_i$ in Task-Order
16.         if existed $vm_m^k$ in vmPool that satisfies constraints
17.             Task-VMC←$vm_m$
18.             VM-TypeC←$vm^k$
19.         elseif existed a new $vm_m^k$ that satisfies constraints
20.             Task-VMC←$vm_m$
21.             VM-TypeC←$vm^k$
22.             add $vm_m^k$ into vmPool
23.         else
24.             $vm_m^k$←the VM that scheduled $t_p$ while $t_p \in pred(t_i)$ and has
                max $TT(e_{pi})$
25.             Task-VMC←$vm_m$
26.             VM-TypeC←$vm^k$
27.         endif
28.     endfor
29.     normalChrom←a chromosome with Task-Order; Task-VMC and
        VM-TypeC
30.     add minMakespanChrom, minCostChrom and normalChrom into
        initPop
31. endwhile
```

Consider the workflow shown in Fig. 1. Each task has a level value l and belongs to the set *BoT(l)*. The set of VM types is {v^0, v^1, v^2, v^3, v^4, v^5}, in which v^0 is the slowest type and v^5 is the fastest. The following process is repeated to generate individuals.

First, tasks are randomly selected from each *BoT* to form a task execution list and *Task-Order* strings, such as 12543768 (lines 3–8).

Second, HEFT is used to generate individuals with the minimum completion time and cost. For the individual with the minimum completion time, we assign tasks on



Index:	1	2	3	4	5	6	7	8
Task-Order:	2	1	4	3	5	7	6	8
Task-VM:	1	5	2	3	6	4	1	2
VM-Type:	3	4	1	3	2	1	3	1

Fig. 3 Encoding of a schedule of the DAG in Fig. 1

the VMs with the fastest speed. The *Task-VM* strings are set to different integers, such as 01234567, while the *VM-Type* strings are set to the fastest VM type, such as 55555555 (lines 9–11). Notably, although this approach does not take into account the data transmission time between different VMs, it can still be considered as an approximate alternative. Similarly, the latter process is to assign all tasks on the same VM with the slowest speed, and *Task-VM* and *VM-Type* are set to 0 (lines 12–14). These two individuals account for only one-third of the total population in order to achieve better optimization performance.

Finally, appropriate VMs are allocated for each task according to the deadline and cost constraints. The VMs are selected according to the following steps: (1) select the VM with the lowest cost among the used VMs while meeting the subdeadline $LFT(t_i)$ of the scheduling task (lines 16–18), (2) create a new VM that has the lowest cost while meeting $LFT(t_i)$ (lines 20–22), and (3) compare the data transfer times $TT(e_{pi})$ between the scheduling task and all its parent tasks and then select the VM on which the parent task that has the maximum value (lines 24–26).

4.4 Fitness evaluation and selection

The fitness of each chromosome is determined by the total execution time *TET* and total execution cost *TEC*. First, tasks are selected and assigned to VMs according to the three strings of a chromosome. The start time *ST* and finish time *FT* of tasks are calculated by Eqs. (7) and (8). Then, the lease start time *LST* and lease finish time *LFT* of the VMs are updated. Finally, the *TET* and *TEC* of the workflow can be calculated according to Eq. (9) after all of the tasks are executed.

Based on the championship mode [11] and constraint processes strategy [40], the select operation between two chromosomes is implemented in the following order: (1) select the individual with the smallest *TEC* when the *TET* of two individuals are both less than the deadline *D*, (2) select the individual that has the *TET* that meets *D* when the *TET* of another exceeds *D*, and (3) select the individual with the smallest *TEC* when the *TET* of two individuals exceeds *D*.

4.5 Crossover

The GA generates offspring through the crossover operation, and the new individuals cannot violate the dependencies between tasks in a workflow. The definitions of *l* and *BoT* are used to implement crossover between the selected parents. The pseudocode is shown in Algorithm 4.

First, a *BoT* is randomly selected between 1 and the number of *BoTs*. Two crossover values, *cross_point* and *cross_length*, are also randomly produced, where *cross_point* plus *cross_length* is less than the number of tasks in the selected *BoT* (lines 2–5). One parent is divided into *Task_Seg1* and *Remain_P1* by the crossover values, and the other parent is divided into *Task_Seg2* and *Remain_P2* by the tasks of *Task_Seg1* (lines 6–9). Subsequently, two candidate individuals are formed according to *Task_Seg2* and *Remain_P1*. The conflicts of different types on the same VMs are resolved by updating the type of VM according to the segments and the remaining parts. The different type in *VM_Type* of one candidate individual is updated to the type in *Task_Seg2* (lines 10–15), while the *VM_Type* of another candidate individual is modified according to *Remain_P1* (lines 16–21). Finally, using the selection method in Section 4.4, the candidate individual with the best fitness is retained and added to the new population (line 22). Another offspring is generated in a similar manner, and the process is omitted in Algorithm 4.

Algorithm 4. Crossover(*P1*, *P2*)

```
1.  randBoT←random selection in the set of BoTs
2.  l←the level of randBoT
3.  n←the length of randBoT
4.  cross_point←random(1, n)
5.  cross_length←random(1, n-cross_point)
6.  Task_Seg1←substring(P1, cross_point, cross_length)
7.  Remain_P1←the remain part of P1
8.  Task_Seg2←substring(P2, cross_point, cross_length)
9.  Remain_P2←the remain part of P2
10. for each vm in Task_Seg2
11.   if vm' type in Remain_P1≠vm' type in Task_Seg2
12.     newRemain_P1←update vm' type in Remain_P1 by the vm'
        type in Task_Seg2
13.   endif
14. endfor
15. Candchild1←a chromosome with Task_Seg2 and newRemain_P1
16. for each vm in Remain_P1
17.   if vm' type in Task_Seg2≠vm' type in Remain_P1
18.     newTask_Seg2←update vm' type in Task_Seg2 by the vm'
        type in Remain_P1
19.   endif
20. endfor
21. Candchild2←a chromosome with newTask_Seg2 and Remain_P1
22. return the better one between Candchild1 and Candchild2
```

Figure 4 is an example of a crossover operation that randomly selects 3 as the level of *BoT*, 1 as the *cross_point*, and 2 as the *cross_length*. After generating the *Task_Segs* according to the parents (grey part), there are type conflicts with v_1 and v_2 between *Task_Seg2* and the rest of *P1*. Thus, two candidate individuals, *Candchild1* and *Candchild2*, are produced. In *VM_Type* of *Candchild1*, the number 3 of index 1 is updated to 4, and the number 1 of index 8 is updated to 3. Similar changes occur in *Candchild2*, and all of the updated values are presented in italics. Finally, *Candchild1* is selected as offspring *Child1* due to its better fitness. Another offspring, *Child2*, is produced in a similar manner.

4.6 Mutation

As in the crossover operation, the constraints between tasks cannot be violated by the mutation operation. Two mutation operations are used to increase the diversity of the solutions. The pseudocode is shown in Algorithm 5.

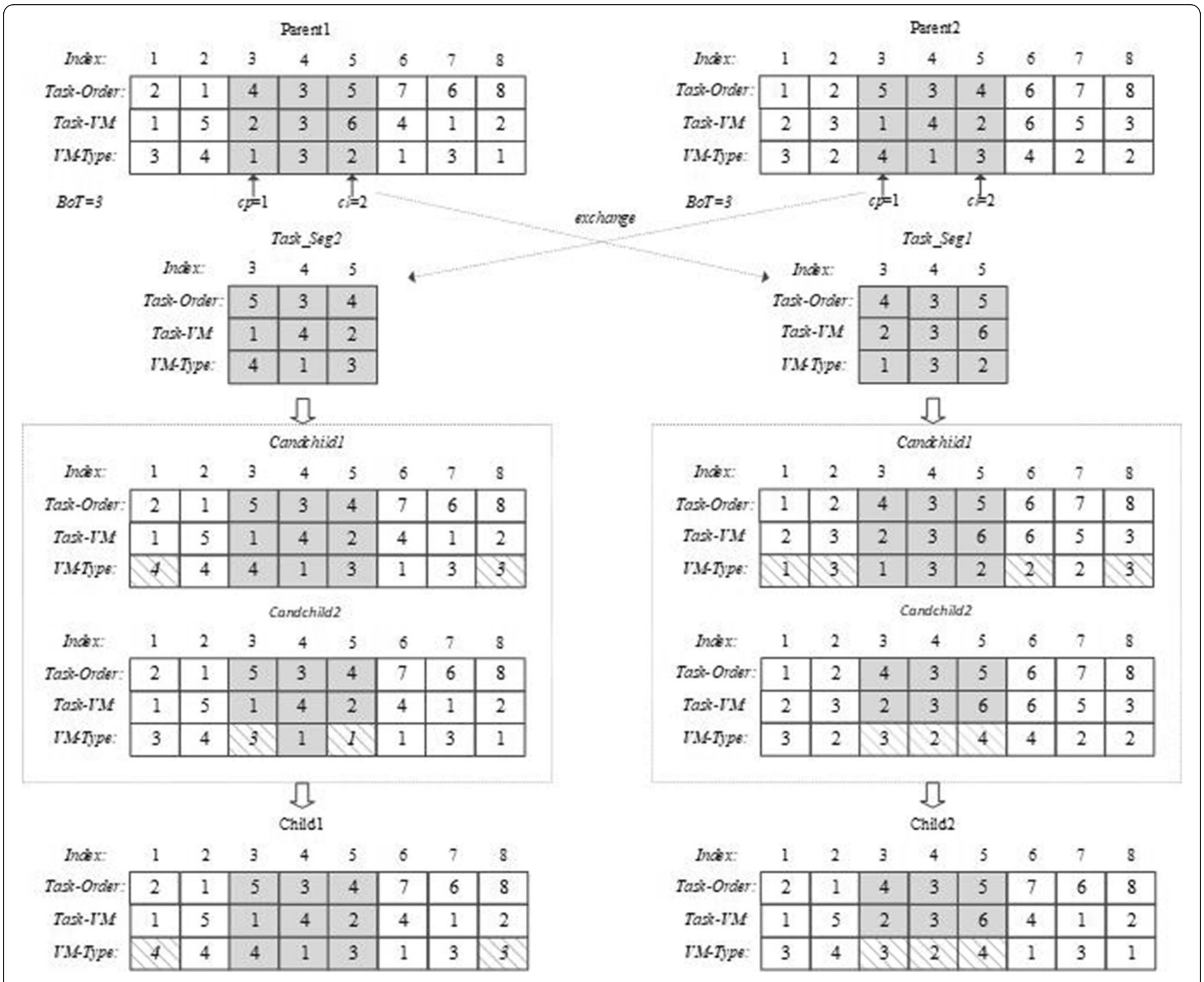


Fig. 4 An example of the crossover operation

```
Algorithm 5. Mutation(X)
1.  l←random(1, LV($t_{entry}$))
2.  $t_i$, $t_r$←randomly select two tasks in BoT(l)
3.  if random(0,1)>0.5
4.      CandChromosome1←swap the value of $t_i$ and $t_r$ in Task_Order
5.      return the better one between CandChromosome1 and X
6.  else
7.      $vm_m^k$←get the least used VM in Task_VM
8.      $vm_s^t$←a random VM in Task_VM
9.      CandChromosome2←update $vm_m^k$ as $vm_s^t$ in Task_VM and
        VM_Type
10.     return the better one between CandChromosome2 and X
11. endif
```

The first mutation operation is the exchange of the task order. Two tasks in the same *BoT* are randomly selected; then, the *Task_Order* is swapped while the values of *Task_VM* and *VM_Type* are unchanged (lines 4–5).

The second mutation operation is the merging of VMs. The least used VM is randomly changed to another used VM. That is, the least frequent number in *Task_VM* is updated to another existing number, and the number in *VM_Type* is correspondingly changed (lines 7–10).

Figure 5 shows an example of the two mutation operations: exchange of task t_4 and t_5 and update of v_5^4 to v_2^1. In the VM merge operation, because several VMs, such as v_5^4, v_3^3, v_6^2, v_4^1, are used only once, v_5^4 is updated to v_2^1 in a random manner.

To avoid the deterioration of individual fitness caused by mutation, the mutated individual and the original individual are compared and the better individual is selected according to the method in Section 4.4.

4.7 Complexity analysis

The overall computational complexity of the proposed algorithm can be analysed in terms of the number of initialization, selection, crossover, mutation, and fitness

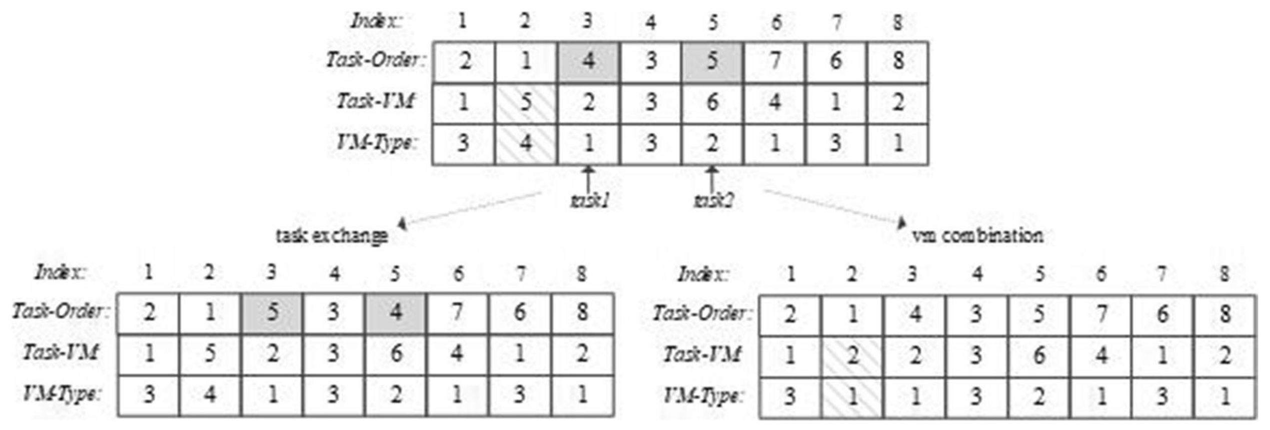


Fig. 5 An example of the mutation operation

evaluation operations. Consider a cloud in which the workflows have n tasks, the number of individuals in the population is p, and the maximum number of iterations is g.

Task initialization is performed by a traversal through each edge and a nested loop of tasks, so the time complexity is $O(n^2)$. Then, for each individual in the initial population, the task selection step has a complexity of $O(n)$. The procedure of mapping each task to an appropriate resource has a maximum complexity of $O(n^2)$ when satisfying the deadline and cost constraints. Therefore, the total time complexity of population initialization is $O(n + n^2)p$. The time complexity of the selection operation is $O(p)$, and the time complexity of the crossover and mutation operations are $O(n^2)$ and $O(n)$, respectively. The fitness evaluation for each individual has a complexity of $O(n^2)$. Finally, the total time complexity of selection, crossover, mutation, and fitness evaluation with p individuals in the population and g iterations is $O(pgn^2)$. Thus, the overall time complexity of DCGA is $O(n^2) + O(n + n^2)p + O(pgn^2)$, which can be expressed as $O(pgn^2)$.

5 Experiments

This section presents the experiments implemented to evaluate the performance of the proposed DCGA.

5.1 Experimental parameters

5.1.1 Workflows

Workflows can achieve service objectives via various construction structures and have the characteristics of low coupling and flexible operation [41]. In order to better simulate IoT applications, the proposed algorithm was evaluated on three real workflows: LIGO, Montage, and Cybershake. These workflows, which have different structures and characteristics, can used to simulate various applications in the IoT environment. The multiple tasks of LIGO require processing large amounts of data; therefore, they can be regarded as data aggregation tasks, which require high computation power. The Montage does not require a large amount of CPU resources, and it focuses on intensive I/O operations. The Cybershake demands high performance computing and has an I/O intensive characteristic. A simplified structure of each workflow is shown in Fig. 6, and a detailed description of these workflows is presented in [42].

Synthetic workflows with similar structures are created by a workflow generator[1]. The generated workflows are represented in the form of a DAG in XML (DAX) format, which contains tasks, dependency edges, transfer data and other information. Then, the DAGs of the experimental workflows can be constructed as XML files.

5.1.2 Baseline algorithms

In our experiments, IC-PCP [23], PSO [29], and a normal GA are used as baseline algorithms to compare with the proposed algorithm.

The IC-PCP algorithm is based on the concept of a partial critical path (PCP), which is defined by unassigned parent tasks and the latest transfer data time. The PCPs related to the exit task are determined recursively, and the tasks on each PCP are assigned to the least expensive available VM or a new VM that can meet the latest start time of each task. Then, the start time and schedule identifier of the tasks are updated. This process is repeated until all tasks in the workflow have been scheduled on the appropriate VMs. Therefore, the cost and makespan of the schedule are calculated based on the lease time of the VMs. However, this algorithm does not consider the characteristics of the VMs, such as acquisition delay and performance variation.

The PSO algorithm is a heuristic optimization method focussed on the deadline constraint for resource allocation. Particles are represented as tasks in the workflow, and their positions represent the allocated resources. Because of the deficiency of VM types in encoding,

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

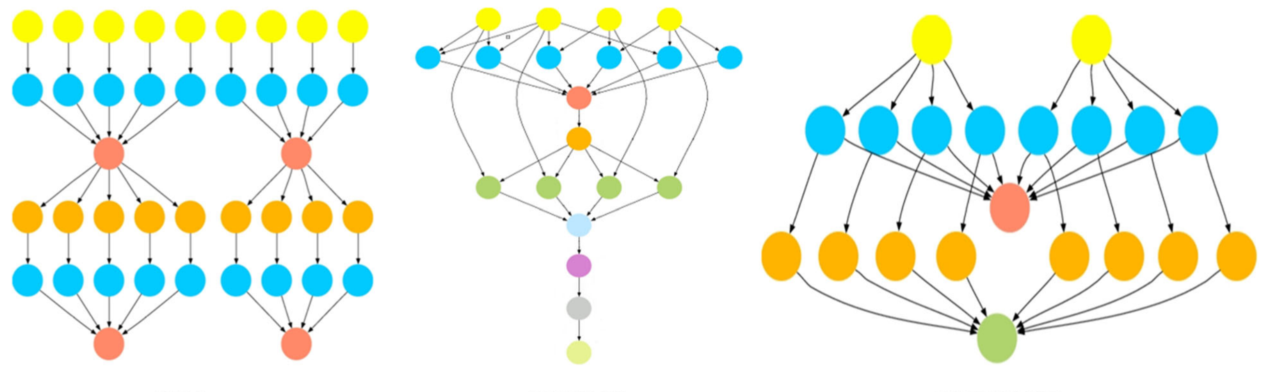


Fig. 6 The structures of the workflows

particles move in different dimensions to the local optimal solution. Furthermore, PSO has difficulty obtaining a feasible solution when the deadline is tight.

The normal GA algorithm (GA_N) is used to analyse the basic characteristics of GAs. The initial population is randomly generated, and roulette wheel selection is used to choose parents. In the crossover phase, tasks with the same level number l are switched between parents. Random mutation is used to create new individuals with different types of VM. The other parameters are similar to the proposed algorithm.

These algorithms were applied in the same simulated cloud environment, together with the DCGA proposed in this paper. The parameters of the PSO algorithm were set according to the experiments in [29].

5.1.3 Experimental setup

The experiments simulated a cloud environment with a single datacenter and eight different VM types. The characteristics of the VM types shown in Table 3 are based on the Amazon cloud services[2]. The average bandwidth between VMs is fixed to 20 MBps, and the transmission cost is assumed to be zero. The processing capacity of each VM type is estimated at one million floating point operations per second (MFLOPS). In a real cloud environment, various factors, such as hardware type, data transmission, and resource allocation, can cause acquisition delay and performance variation of VMs. Therefore, an acquisition delay of 97 s, as presented in [43], and the performance of VMs, which was diminished by at most 24% based on a normal distribution with a mean of 12% and a standard deviation of 10%, as presented in [10], are adopted in our experiments.

To investigate the performance of these algorithms under different deadlines, the method mentioned in Section 4 is used to obtain the benchmark deadline D_B. D_B can be approximately calculated as the finish time of task t_{exit} when all tasks are allocated to different VMs of the fastest type. The following formula is used to obtain different deadlines:

$$D = \alpha * D_B, 1 \le \alpha \le 4 \tag{12}$$

where α is the deadline factor ranging from 1 to 4 with a step size of 0.2. Three different levels of deadlines can be classified as follows: (1) strict deadlines ($1 \le \alpha < 2$), (2) moderate deadlines ($2 \le \alpha < 3$), and (3) relaxed deadlines ($3 \le \alpha \le 4$).

The parameters of the proposed DCGA are as follows: the population size is 500, the maximum number of generations is 100, the crossover rate is 80%, and the mutation rate is 20%. Initially, 500 solutions are created by the methods presented in Sections 4.1 and 4.3. Then, these initial chromosomes are evaluated, and the next population is generated by selection, crossover, mutation, and fitness evaluation. Finally, the feasible solution with the minimum *TEC* under the deadline constraint D in every generation is found.

5.2 Results and analysis

Under the different deadline constraints for the three workflows, each experiment is executed ten times, and

Table 3 VM types used in the experiments

Type	Compute unit	Price ($)
t2.small	3.3	0.023
t2.medium	6.6	0.0464
t2.xlarge	12	0.1856
m5.2xlarge	20	0.384
m5.4xlarge	40	0.768
m5.12xlarge	120	2.304
m4.16xlarge	147.2	3.20
m5.24xlarge	240	4.608

[2]https://aws.amazon.com/cn/ec2/pricing/

the mean of the results is used for analysis. According to the deadline and cost constraints, the experimental analyses are carried out with respect to three indicators: success rate, execution makespan, and execution cost. The definitions of these indicators are as follows:

1. Success rate: the ratio between the number of simulations runs that satisfy the constraints and the total number of simulations runs. The success rate represents the effectiveness of the algorithm.
2. Execution time: the finish time of the last task t_{exit} in a schedule, which can be calculated by Eq. (8). Usually, a shorter execution time means better performance of the algorithm.
3. Execution cost: the lease cost of all VMs used in a schedule, which can be calculated by Eq. (9). The execution cost of the algorithm can only be meaningfully compared under the deadline constraint.

5.2.1 Success rate

Table 4 shows the success rates of all the algorithms under different deadline constraints. For the strict deadline constraint, the success rate of IC-PCP is the lowest, which means that the algorithm fails to obtain a feasible solution in most cases. GA_N achieves a slight improvement over IC-PCP on the Cybershake workflow. PSO achieves a certain success rate for the strict deadline constraint, with success rates of 72% on LIGO, 78% on Montage, and 80% on Cybershake. With a mean success rate of 78%, DCGA achieves the best performance. In terms of the success rate, for LIGO, DCGA outperforms GA_N and IC-PCP by 74% and outperforms PSO by 2%. For Montage, DCGA outperforms GA_N by 6% and outperforms IC-PCP and PSO by 2%. For Cybershake, DCGA outperforms IC-PCP by 80% and outperforms GA_N by 8%.

Table 4 Total success rates for workflows

Deadline constraint	Algorithms	LIGO	Montage	Cybershake
Strict deadline	GA_N	0	74	72
	IC-PCP	0	78	0
	PSO	72	78	80
	DCGA	74	80	80
Moderate deadline	GA_N	22	100	100
	IC-PCP	0	100	98
	PSO	98	100	100
	DCGA	100	100	100
Relaxed deadline	GA_N	80	100	100
	IC-PCP	45	100	100
	PSO	100	100	100
	DCGA	100	100	100

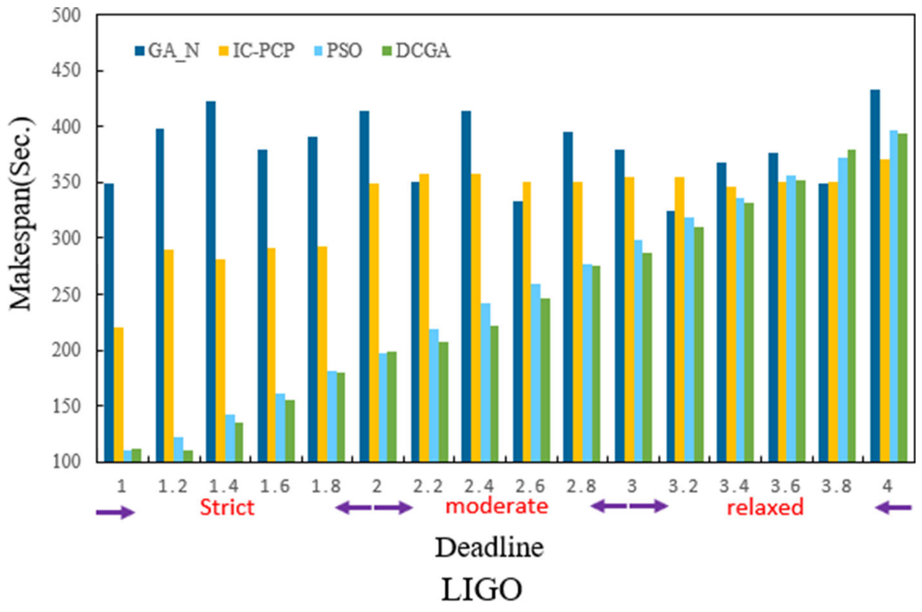
The success rates of these algorithms are greatly improved when the deadline is moderate or relaxed. Under a moderate deadline, the success rate of GA_N is 22%, while IC-PCP fails to obtain a feasible solution on LIGO. The average success rate of PSO is increased to 99.3%, and that of DCGA is 100%. For the relaxed deadline constraint on LIGO, DCGA outperforms GA_N by 20% and outperforms IC-PCP by 55%. All four algorithms achieve their highest success rate on Montage and Cybershake.

The results in Table 4 show that IC-PCP gives poor performance, while DCGA gives the best performance for each deadline constraint. Because IC-PCP does not consider the performance variation and acquisition delay of VMs, it is easy to prolong the finish time of tasks, which will cause the makespan of workflow to exceed the deadline. Because of the instability and randomness of the original algorithm, GA_N also has poor performance compared to that of PSO and DCGA. PSO ignores more information of the VM on particle coding and fails to exploit the effect of VM type on task execution cost, so the cost to obtain a higher success rate is increased. The DCGA proposed in this paper takes into account the main characteristics of the cloud environment and searches for the best solution through encoding, selection, crossover and mutation; therefore, its performance is superior to that of the other three algorithms.

5.2.2 Makespan and cost evaluation

To obtain more details about the performance of all the algorithms, the average execution time and average execution cost are compared under strict, moderate, and relaxed deadline constraints. Figures 7, 8 and 9 show the average makespan and average cost under these deadline factors for different workflows.

1. Figure 7 shows the average makespan and average cost results achieved on LIGO. With a long makespan in most cases, GA_N and IC-PCP fail to generate a feasible solution under strict and moderate deadline constraints. GA_N has the worst performance with the highest cost, while the performance of IC-PCP is slightly better. Although PSO is the least expensive, the algorithm does not obtain valid solutions without violating the deadline constraints. The proposed DCGA has low execution cost under different deadline constraints: the average cost is 89.83% lower than that of GA_N, 38.8% higher than that of IC-PCP, and 60.99% lower than that of PSO. Thus, DCGA produces effective feasible solutions and achieves the best performance with a smaller cost than that of the other algorithms on the LIGO workflow.



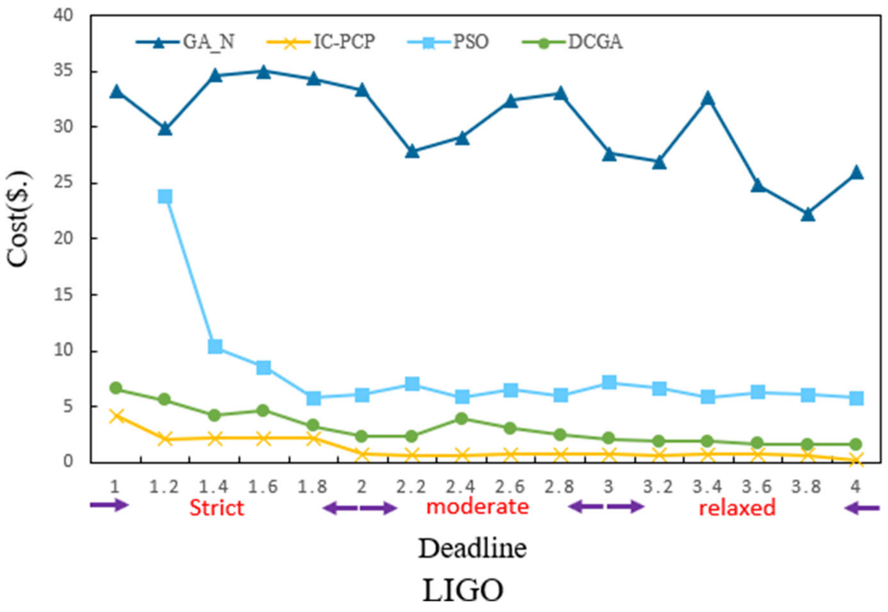


Fig. 7 Makespan and cost of scheduling on LIGO under different deadline constraints

2. The simulation results for the Montage workflow are shown in Fig. 8. The average execution time of the schedules generated by GA_N and IC-PCP is less than that of PSO and DCGA. The average cost of GA_N is the highest, while DCGA has the lowest cost. The performance of PSO is slightly improved compared to LIGO. However, PSO fails to obtain a feasible solution when the deadline factor is 1, so the cost is not shown in the figure. The DCGA algorithm achieves better average execution cost than that of IC-PCP but has a larger makespan. For the Montage workflow under moderate deadline constraints, DCGA obtains an average cost that is 98.45% lower than that of GA_N, 52.07% lower than that of IC-PCP, and 92.16% lower than that of PSO. The average execution cost of schedules generated by the proposed algorithm is minimized by using a large execution time that satisfies the deadline constraint.
3. The average makespan and average cost results obtained by the tested algorithms on Cybershake are shown in Fig. 9. IC-PCP failed to obtain valid solutions meeting the strict deadline constraint, while GA_N slightly improved its performance. Meanwhile, IC-PCP cannot obtain a feasible solution when the deadline factor is less than 1.6, so the cost

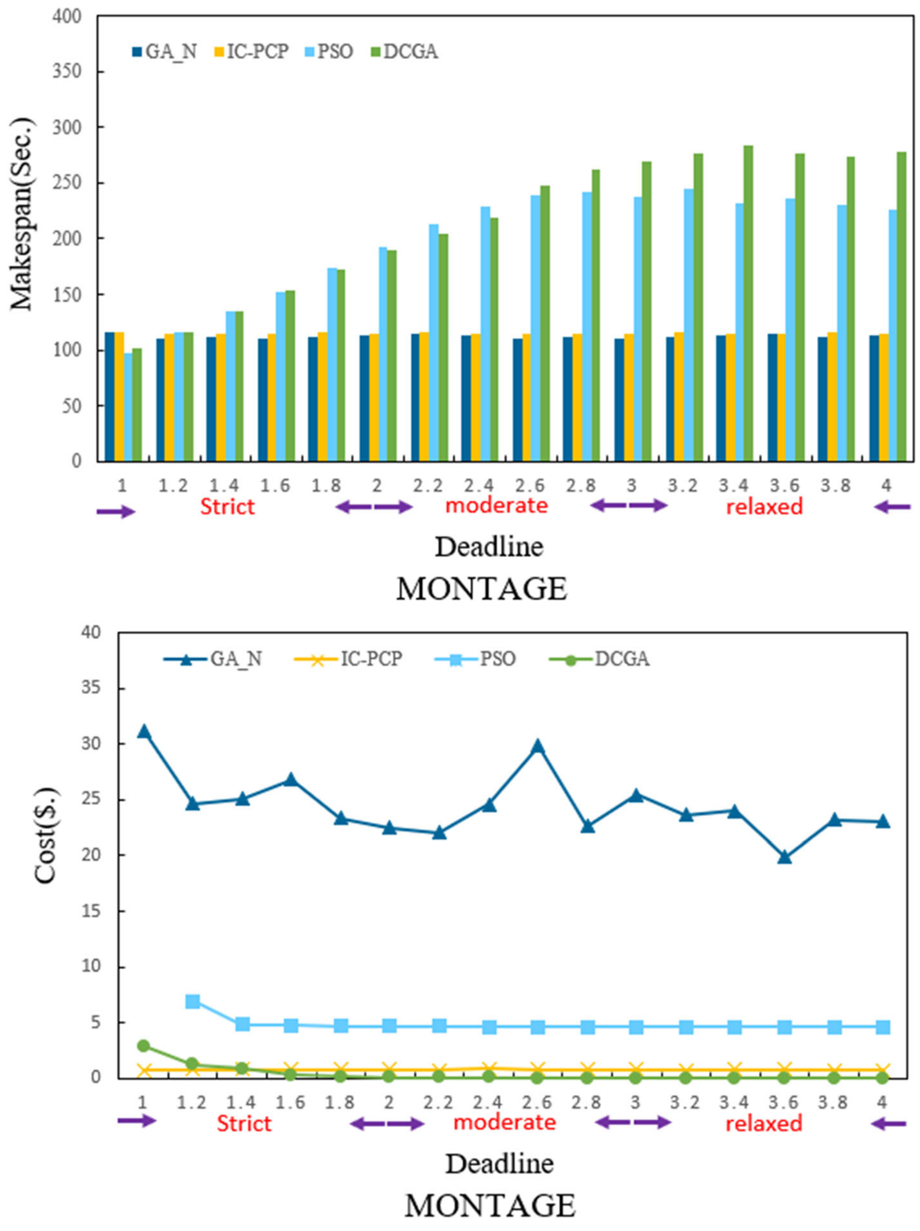


Fig. 8 Makespan and cost of scheduling on MONTAGE under different deadline constraints

is not shown in the figure. The same situation occurs for PSO when the deadline factor is 1. The cost of the proposed DCGA is 98.02% lower than that of GA_N, 72.96% lower than that of IC-PCP, and 89.87% lower than that of PSO. Among the compared algorithms, DCGA always achieves the minimum execution cost while meeting the deadline constraints.

5.3 Summary of scheduling algorithms performance

In general, the DCGA proposed in this paper can obtain feasible solutions under different deadline constraints. The proposed algorithm not only has a high success rate but can also achieve the minimum execution cost under deadline constraints. In particular, Figs. 7, 8, and 9 show that DCGA performs better than the baseline algorithms in terms of makespan and cost regardless of whether the workflow is I/O intensive or computationally complex. This good performance is due mainly to the following reasons:

1. GA_N shows only the characteristics of GA without considering the optimization of the population initialization, selection, crossover, and other

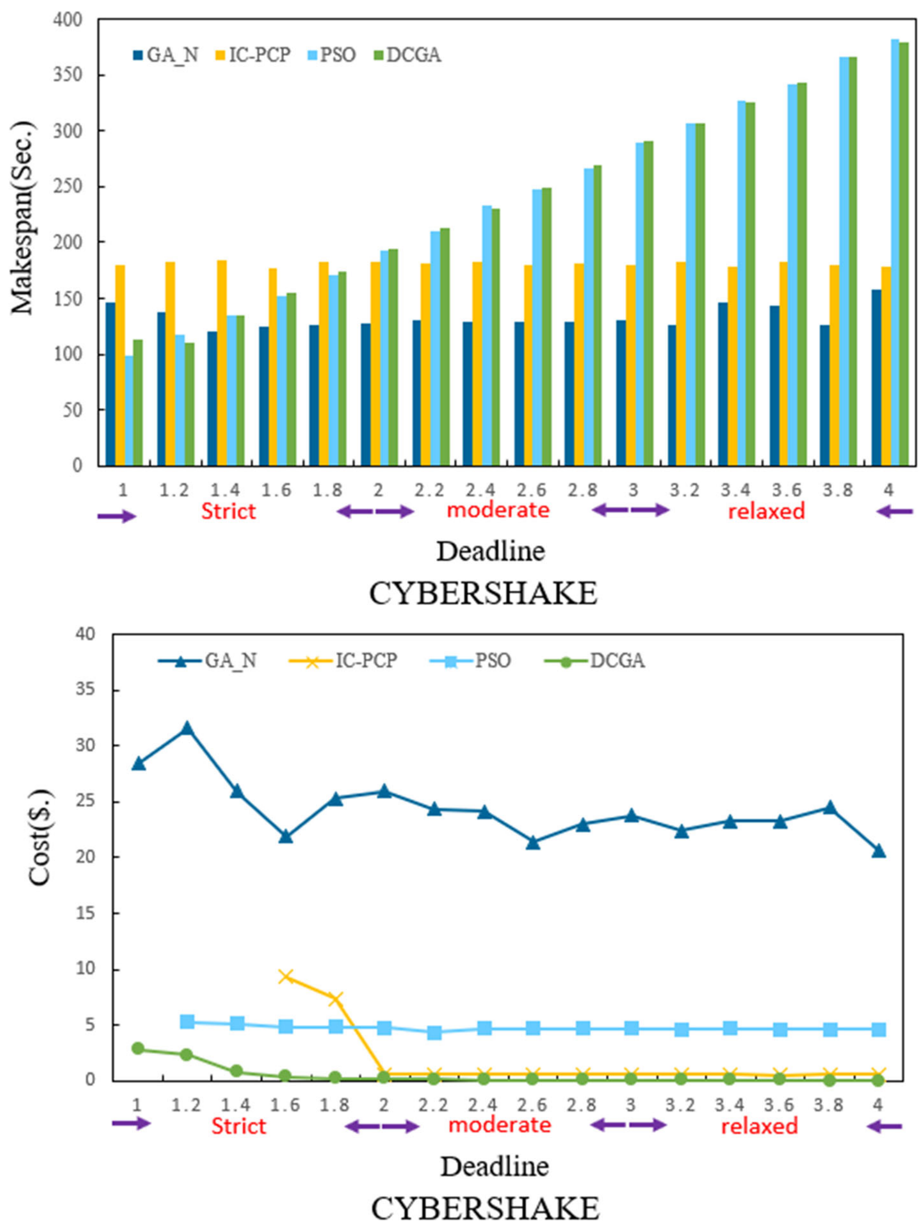


Fig. 9 Makespan and cost of scheduling on Cybershake under different deadline constraints

operations. Because the initial population is randomly generated, the search space is too large to converge to the optimal solution. Although the algorithm can sometimes obtain solutions that meet certain conditions, the average execution cost is always much higher than that of the other algorithms because of the randomness and instability.

2. IC-PCP fails to take into account the actual characteristics of the cloud environment. When a task is assigned to a VM, the performance variation and acquisition delay of VMs affect not only the task and other tasks in the critical path but also tasks on other critical paths of the workflow. Thus, a missed subdeadline for a task extends the makespan of the entire workflow, so the algorithm fails to produce feasible solutions under the strict deadline.
3. PSO does not provide sufficient VM information through the encoding of particles. When the velocity and position of particles are updated, the insufficient consideration of VM types causes the particles to tend toward the local optimum rather than the global optimum. When the deadline is tight, feasible solutions are difficult to obtain, which

negatively affects the success rate of the algorithm. Under moderate and relaxed deadline constraints, PSO gives less consideration to the cost of different VM types, thus affecting actual performance.

4. The proposed DCGA focuses on the main features in the cloud environment. The matching of tasks and VMs can be reflected by the appropriate coding of chromosomes, which has good adaptability. Due to the effective use of coding features, individuals with better makespan and cost can be inherited to generate the final optimized solution.

Therefore, DCGA has the best performance in terms of producing cost-effective and feasible schedules under deadline constraints.

6 Conclusions and further work

In this paper, the scheduling problem of the workflow of IoT applications is studied, and a deadline and cost-aware genetic optimization algorithm is proposed. With the purpose of minimizing the cost under a deadline constraint, the proposed algorithm focuses on the important features of the cloud, such as on-demand acquisition, heterogeneous dynamics, acquisition delay, and performance variation of VMs. Furthermore, the proposed algorithm uses heuristic operations to assign tasks to appropriate VMs. The experimental results demonstrate that among state-of-art algorithms, our algorithm not only exhibits the highest success rate but also achieves low execution costs under various deadline constraints.

In the future, we would like to consider other issues, such as task reassignment and VM failure, because these factors will affect the stability and reliability of the proposed algorithm. The cost of communication among VMs in a real cloud environment must also be considered. Furthermore, we intend to update the design of the population initialization, encoding, and crossover of the proposed algorithm to improve the performance and efficiency.

Abbreviations

ACO: Ant colony optimization; BDAS: Budget deadline aware scheduling; BoT: Bag of tasks; DAG: Directed acyclic graphs; DAX: Directed acyclic graph in XML; DBL: Deadline bottom level; DCGA: Deadline and cost-aware genetic algorithm; DFS: Depth-first search; DTL: Deadline top level; ECMS-MOO: Endocrine-based coevolutionary multi-swarm for multi-objective optimization; GA: Genetic algorithm; GA_N: The normal GA algorithm; IaaS: Infrastructure as a service; IC-PCP: IaaS cloud partial critical path; IoT: Internet of Things; JIT-C: Just in time; MFLOPS: Million floating point operations per second; PCP: Partial critical paths; PSO: Particle swarm optimization; RCT: Robustness-cost-time; RTC: Robustness-time-cost; SCS: Scaling-consolidation-scheduling; VM: Virtual machine

Acknowledgements

We want to thank the authors of the literature cited in this paper for contributing useful ideas to this study.

Authors' contributions

XM came up with the initial concept of DCGA, the system design, and implemented the prototype. He wrote the majority of this paper. HG participated in the design process, provided feedback, and helped to improve the writing. HX and MB both supported the case study with their deep knowledge about workflow scheduling in cloud computing. All authors read and approved the final manuscript.

Authors' information

Xiaojin Ma received the BS, MS in Computer Science and Management Science & Engineering from Henan University of Science and Technology in 2003, 2013, respectively. He is working toward the Ph.D. degree in Shanghai University, China.
Honghao Gao received the Ph.D. degree in Computer Science and started his academic career at Shanghai University in 2012. He is an IET Fellow, BCS Fellow, EAI Fellow, IEEE Senior Member, CCF Senior Member, and CAAI Senior Member. Prof. Gao is currently a Distinguished Professor with the Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, China. His research interests include service computing, model checking-based software verification, and sensors data application.
Huahu XU received the Ph.D. degree in Computer Application from Shanghai University. He is currently a Doctoral Supervisor and a Professor with the School of Computer Engineering and Science, Shanghai University, where he is also the Director of the Information Office of Shanghai University. He is the Chairman of the Shanghai Security and Technology Association.
Minjie Bian received the Ph.D. degree in Computer Application Technology from the School of Computer Engineering and Science, Shanghai University, Shanghai, China, in 2016.

Funding

The work in this paper was supported by National Natural Science Foundation of China (NFSC) under Grant No. 61902236, and CERNET Innovation Project under Grant No. NGII20170513, NGII20170206, NGII20180503, NGII20180617.

Availability of data and materials

The datasets of all our measurements analysed in this study are available in the following repositories:
1. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator
2. https://aws.amazon.com/cn/ec2/pricing/

Competing interests

The authors declare that they have no competing interests.

Author details

[1]School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China. [2]School of Management, Henan University of Science and Technology, Luoyang 471000, China. [3]Computing Center, Shanghai University, Shanghai 200444, China. [4]Shanghai Shangda Hairun Information System Co., Ltd, Shanghai 200444, China.

Received: 3 July 2019 Accepted: 11 September 2019
Published online: 08 November 2019

References

1. V. Mayer-Schonberger, K. Cukier, Big data: a revolution that will transform how we live, work, and think. Mathematics & Computer Education **47**(17), 181–183 (2014). https://doi.org/10.1093/aje/kwu085
2. T. Qiu, N. Chen, K. Li, et al., How can heterogeneous internet of things build our future: a survey. IEEE Communications Surveys & Tutorials **20**(3), 2011–2027 (2018). https://doi.org/10.1109/COMST.2018.2803740
3. J. Pan, J. McElhannon, Future edge cloud and edge computing for internet of things applications. IEEE Internet of Things Journal **5**(1), 439–449 (2018). https://doi.org/10.1109/JIOT.2017.2767608
4. P. Mell, The NIST definition of cloud computing. Communications of the ACM **53**(6), 50 (2011). https://doi.org/10.6028/NIST.SP.800-145
5. S. Aizad, A. Anjum, R. Sakellariou, "Representing variant calling format as directed acyclic graphs to enable the use of cloud computing for efficient and cost effective genome analysis," 2017 17th IEEE/ACM International

Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 784-787, Madrid, 2017. doi: https://doi.org/10.1109/CCGRID.2017.116

6. T. Sousa, A. Silva, A. Neves, Particle swarm based data mining algorithms for classification tasks. Parallel Comput. **30**(5–6), 767–783 (2004). https://doi.org/10.1016/j.parco.2003.12.015
7. F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: a survey. J. Supercomput. **71**(9), 3373–3418 (2015). https://doi.org/10.1007/s11227-015-1438-4
8. H.H. Gao, W.Q. Huang, Y.C. Duan, et al., Research on cost-driven services composition in an uncertain environment. Journal of Internet Technology **20**(3), 755–769 (2019). https://doi.org/10.3966/160792642019052003009
9. K.R. Jackson, L. Ramakrishnan, K. Muriki, et al., in *2010 IEEE Second International Conference on Cloud Computing Technology & Science*. Performance analysis of high performance computing applications on the Amazon Web services cloud (Indianapolis, 2010). https://doi.org/10.1109/CloudCom.2010.69
10. J. Schad, J. Dittrich, J.-A. Quiané-Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proceedings of VLDB Endowment **3**(1), 460–471 (2010). https://doi.org/10.14778/1920841.1920902
11. D. Whitley, A genetic algorithm tutorial. Stat. Comput. **4**(2), 65–85 (1994). https://doi.org/10.1007/BF00175354
12. R. Calheiros, R. Buyya, Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Transactions on Parallel & Distributed Systems **25**(7), 1787–1796 (2014). https://doi.org/10.1109/TPDS.2013.238
13. R. Chard, K. Chard, K. Bubendorfer, et al., in *IEEE International Conference on E-science*. Cost-aware cloud provisioning (2015). https://doi.org/10.1109/eScience.2015.67
14. J. Yu, R. Buyya, C.K. Tham, Cost-based scheduling of scientific workflow application on utility grids. Proc.1st Int.Conf. e-Sci.Grid Comput.e-Sci, 140–147 (2005). https://doi.org/10.1109/E-SCIENCE.2005.26
15. J. Yu, M. Kirley, R. Buyya, in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, GRID'07*. Multi-objective planning for workflow execution on grids (IEEE Computer Society, Washington, 2007), pp. 10–17. https://doi.org/10.1109/GRID.2007.4354110
16. R. Sakellariou, H. Zhao, E. Tsiakkouri, M. D. Dikaiakos, "Scheduling workflows with budget constraints," Integrated Research in Grid Computing, S. Gorlatch and M. Danelutto, Eds., CoreGrid series. New York, NY, USA: Springer-Verlag, pp. 189-202, 2007. doi: https://doi.org/10.1007/978-0-387-47658-2_14
17. R. Duan, R. Prodan, T. Fahringer, Performance and cost optimization for multiple large-scale grid workflow applications. IEEE Conference on Supercomputing, 112 (2007). https://doi.org/10.1109/TSMCC.2008.2001722
18. W.N. Chen, J. Zhang, An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **39**(1), 29–43 (2009). https://doi.org/10.1109/tsmcc.2008.2001722
19. C. Hoffa, G. Mehta, T. Freeman, et al., On the use of cloud computing for scientific workflows. Fourth IEEE International Conference on eScience, e-Science, 640–645 (2008). https://doi.org/10.1109/eScience.2008.167
20. E.-K. Byun, Y.-S. Kee, J.-S. Kim, et al., Cost optimized provisioning of elastic resources for application workflows. Futur. Gener. Comput. Syst. **27**(8), 1011–1026 (2011). https://doi.org/10.1016/j.future.2011.05.001
21. E.-K. Byun, Y.-S. Kee, J.-S. Kim, et al., BTS: resource capacity estimate for time-targeted science workflows. J. Parallel Distrib. Comput. **71**(6), 848–862 (2011). https://doi.org/10.1016/j.jpdc.2011.01.008
22. N. Malawski, G. Juve, E. Deelman, et al. "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in Proc. Int. Conf. High Perform. Comput. Netw., Storage Anal., Art. no. 22, 2012. doi: https://doi.org/10.1109/SC.2012.38
23. S. Abrishami, M. Naghibzadeh, D.H.J. Epema, Deadline constrained workflow scheduling algorithms for infrastructure as a service clouds. Futur. Gener. Comput. Syst. **29**(1), 158–169 (2013). https://doi.org/10.1016/j.future.2012.05.004
24. S. Pandey, L. Wu, S.M. Guru, et al., in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments (AINA, 2010), pp. 400–407. https://doi.org/10.1109/AINA.2010.31
25. G. Yao, Y.S. Ding, Y.C. Jin, et al., Endocrine-based coevolutionary multi-swarm for multi-objective workflow scheduling in a cloud system. Soft. Comput. **21**, 4309–4322 (2017). https://doi.org/10.1007/s00500-016-2063-8
26. Q. Wu, G.H. Qin, B.B. Huang, The research of multimedia cloud computing platform data dynamic task scheduling optimization method in multi core environment. Multimed Tools Appl **76**, 17163–17178, 201. https://doi.org/10.1007/s11042-016-3667-9
27. Q.W. Wu, F. Ishikawa, Q.S. Zhu, et al., Deadline-constrained cost optimization approaches for workflow scheduling in clouds. IEEE Transactions on Parallel and Distributed Systems **28**(12), 3401–3412 (2017). https://doi.org/10.1109/TPDS.2017.2735400
28. M. Mao, M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," Conference on High Performance Computing Networking, Storage Anal., Art. no. 49, 2011. doi: https://doi.org/10.1145/2063384.2063449
29. M.A. Rodriguez, R. Buyya, Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Transactions on Cloud Computing **2**(2), 222–235 (2014). https://doi.org/10.1109/TCC.2014.2314655
30. D. Poola, S.K. Garg, R. Buyya, et al., in *IEEE International Conference on Advanced Information Networking & Applications*. Robust scheduling of scientific workflows with deadline and budget constraints in clouds (IEEE, 2014). https://doi.org/10.1109/AINA.2014.105
31. J. Sahni, P. Vidyarthi, A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Transactions on Cloud Computing **6**(1), 2–18 (2018). https://doi.org/10.1109/TCC.2015.2451649
32. V. Arabnejad, K. Bubendorfer, B. Ng, Budget and deadline aware e-science workflow scheduling in clouds. IEEE Transactions on Parallel and Distributed Systems **30**(1), 29–44 (2019). https://doi.org/10.1109/TPDS.2018.2849396
33. Z.G. Chen, K.J. Du, Z.H. Zhan, et al., Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm. Proc. IEEE Congr. Evol. Com-put, 708–714 (2015). https://doi.org/10.1109/CEC.2015.7256960
34. Z.M. Zhu, G.X. Zhang, M.Q. Li, et al., Evolutionary multi-objective workflow scheduling in cloud. IEEE Transactions on Parallel & Distributed Systems **27**(5), 1344–1357 (2016). https://doi.org/10.1109/TPDS.2015.2446459
35. J. Meena, M. Kumar, M. Vardham, Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint. IEEE Access **4**, 5065–5082 (2016). https://doi.org/10.1109/ACCESS.2016.2593903
36. W.L. Li, Y.N. Xia, M.C. Zhou, et al., Fluctuation-aware and predictive workflow scheduling in cost-effective infrastructure-as-a-service clouds. IEEE Access **6**, 61488–61502 (2018). https://doi.org/10.1109/ACCESS.2018.2869827
37. L.Y. Qi, Y. Chen, Y. Yuan, et al., A QoS-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems. World Wide Web Journal (2019). https://doi.org/10.1007/s11280-019-00684-y
38. Tarjan, "Depth-first search and linear graph algorithms," Symposium on Switching & Automata Theory. IEEE, 2008. doi: https://doi.org/10.1109/SWAT.1971.10
39. Y.C. Yuan, X.P. LI, Q. Wang, et al., Bottom level based heuristic for workflow scheduling in grids. Chines Journal of Computers **31**(2), 282–290 (2008). https://doi.org/10.3321/j.issn:0254-4164.2008.02.012
40. K. Deb, A. Pratap, S. Agarwal, et al., A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
41. H.H. Gao, W.Q. Huang, X.X. Yang, et al., Towards service selection for workflow reconfiguration: an interface-based computing. Future Generation Computer Systems **87**, 298–311 (2018). https://doi.org/10.1016/j.future.2018.04.064
42. G. Juve, A. Chervenak, E. Deelman, et al., Characterizing and profiling scientific workflows. Futur. Gener. Comput. Syst. **29**(3), 682–692 (2013). https://doi.org/10.1016/j.future.2012.08.015
43. M. Mao, M. Humphrey, A performance study on the VM startup time in the cloud. IEEE Fifth International Conference on Cloud Computing, 423–430 (2012). https://doi.org/10.1109/CLOUD.2012.103