Cluster Computing (2022) 25:3265–3281
https://doi.org/10.1007/s10586-022-03561-y

A two-stage scheduling method for deadline-constrained task in cloud computing

Xiaojian He[1] · Junmin Shen[1] · Fagui Liu[1] · Bin Wang[2] · Guoxiang Zhong[1] · Jun Jiang[1]

Received: 6 November 2021 / Revised: 21 January 2022 / Accepted: 2 February 2022 / Published online: 18 February 2022

Abstract
In a cloud environment, reducing energy consumption while ensuring diverse quality of service (QoS) guarantees is challenging for task schedulers. Specifically, the energy-efficient scheduling for real-time tasks is more complicated because such tasks have strict time constraints. In this paper, we propose a two-stage scheduling method for deadline-constrained tasks. In the first stage, Enhanced Ant Colony Optimization (EACO) is a global scheduler that allocates incoming cloud tasks to suitable virtual machines (VMs). It can minimize makespan and energy consumption while guaranteeing strict deadline constraints. In the second stage, the Modified Backfilling (MBF) algorithm reorders VM's waiting queue to improve the task completion rate. We conduct two experiment series on synthetic and real trace datasets using the Cloudsim toolkit. Extensive experiments show that compared with other well-known task scheduling methods, our method can effectively reduce makespan by 25.28% and energy consumption by 23% on average. The task completion rate can be increased by 6.27%. The proposed method has a significant improvement compared with other well-known algorithms.

Keywords Cloud computing · Task scheduling · Deadline · Ant colony optimization · Energy consumption

1 Introduction

Cloud computing abstracts tremendous servers into virtual resources through virtualization technology and provides users with a pay-per-use fashion [1, 2]. This paradigm had significantly reduced the financial and maintenance cost of acquiring computing resources for application deployment. Real-time applications deployed on the cloud have strict time constraints [3], such as multimedia, video streaming, internet of things (IoT), etc. Backfilling algorithms [4] are widely used in cloud computing to improve resource utilization and task response time. It discovers idle time slots between tasks and inserts suitable tasks in this gap, thereby improving resource utilization and task waiting time [5]. This method can significantly improve task completion rate but neglect the energy consumption.

With the development of cloud computing, energy consumption has become a major problem that can not be ignored [6]. Reducing energy consumption has become a hot spot that most researchers have devoted to in recent years [7]. High energy consumption in cloud data centers increases carbon emissions and affects the reliability of the system [8]. This can be supported by Arrhenius's life-stress model [9], which believes that "for every 10$^\circ C$ increase in temperature, the failure rate of electronic devices will increase by a factor of two". Therefore, it is necessary to take measures to reduce energy consumption in the cloud data center. Virtual machine (VM) consolidation [10] is the main method to reduce energy consumption in the cloud data center. It takes hot migration technology to consolidate low-utilization VMs and switches idle servers to

✉ Fagui Liu
fgliu@scut.edu.cn

✉ Bin Wang
wangb02@pcl.ac.cn

Xiaojian He
hexj@scut.edu.cn

Guoxiang Zhong
cszhongguoxiang111@mail.scut.edu.cn

1 School of Computer Science and Engineering, South China University of Technology, Guangzhou, China

2 Cyberspace Security Research Center, Peng Cheng Laboratory, ShenZhen, China

energy-saving mode [11, 12]. Although VM consolidation can significantly reduce energy consumption, it will also generate communication costs in VM migration. If we consider energy consumption and load balance in the task scheduling phase, the communication costs will decrease as VM consolidation triggering decreases.

Task scheduling is a vital step in cloud computing to improve the performance of the cloud system and ensure quality of service (QoS) [13, 14]. An efficient scheduling algorithm can not only reduce the energy consumption of the data center, improve resource utilization, but also improve the response time of the task [15, 16]. However, task scheduling problems are still NP-complete [17]. When the scale of the problem increases, the algorithm will suffer dimension explosion [18]. Hence, most algorithms are searching for an acceptable sub-optimal solution. In recent years, meta-heuristic algorithms and their variations have been widely used in cloud computing to solve scheduling problems [19]. It may benefit from the high efficiency of the meta-heuristic algorithm and can find an approximate optimal solution in a polynomial time [2], include ant colony optimization (ACO) [20, 21], genetic algorithm (GA) [22, 23], particle swarm optimization (PSO) [24, 25], etc.

There are two kinds of problems we face during the scheduling phase. The first one is when a relatively larger task is assigned to a VM with weak processing ability. We can reduce energy consumption, but the task processing time will increase, which may cause the task can not complete before its deadline. The second one is that when a small task allocates to a VM with strong processing ability, it will consume more energy and make a large task stay waiting for extra time. This can be supported by [26] energy model. In other words, we should schedule tasks on the most suitable VMs to trade-off several conflict metrics. There are lot of work devoted to multi-objective optimization scheduling problems and mainly optimize cost, energy consumption, and response time [27]. However, these methods can not achieve a good balance between optimizing energy consumption and task completion rate under deadline constraints.

To resolve the mentioned issues, this paper proposes a two-stage scheduling method for deadline-constrained tasks in cloud computing. In the first phase, Enhanced Ant Colony Optimization (EACO) is a global scheduler that allocates incoming cloud tasks to suitable VMs. This stage can minimize makespan and energy consumption while satisfying task response time. In the second phase, the Modified Backfilling (MBF) algorithm reorders the VM's waiting queue to improve the task completion rate. We conduct simulation experiments using the CloudSim toolkit to verify the effectiveness of our algorithm. The main contributions of this paper are as follows:

1. We propose a two-stage task scheduling framework based on the ant colony algorithm and backfilling strategy to solve and optimize the task scheduling scheme by considering the makespan and energy.
2. We present a novel ACO based on deadline constraints to obtain a near-optimal task scheduling scheme to balance energy consumption and task completion rate. In addition, according to the sensitivity of the task deadline, the MBF strategy is proposed to adjust the execution order of tasks in the VMs waiting queue to improve the task completion rate further.
3. Experimental results show that compared with other well-known methods, our algorithm can better guarantee the timely response of deadline-constrained tasks and minimize the energy consumption of cloud systems.

The rest of this paper is organized as follows: Sect. 2 introduces related work. Section 3 presents our proposed scheduling framework and the models used in this paper. We introduce the implementation detail of EACO–MBF in Sect. 4. Section 5 evaluates the performance of the proposed algorithms. Finally, we conclude this paper in Sect. 6.

2 Related works

The responsibility of the cloud task scheduler is to allocate resources reasonably so that tasks can be executed on appropriate computing resources. In this process, it is necessary to guarantee the QoS of users and maximize the benefits of cloud service providers. Different users have different requirements for QoS, such as cost, response time, and reliability [28]. Therefore, service profit and energy consumption are significant issues that cloud providers need to consider urgently. Hence, this conflict problem is what the scheduler needs to address. Moreover, the dynamics and heterogeneity of the cloud environment further complex this problem [29]. There exists numerous work in task optimize scheduling in the cloud environment, including energy-aware and QoS-aware. We classify these work into two categories: (i) Energy-aware task scheduling, and (ii) Multi-objective task scheduling.

2.1 Energy-aware task scheduling

Task scheduling is a vital step to reduce energy consumption in cloud data centers. Efficient task scheduling not only improves system performance but also cuts down energy consumption.

Yuan et al. [30] used the G/G/1 queuing model to analyze server performance and proposed a honeybee algorithm based on simulated annealing to achieve task energy optimization scheduling. Taking into account energy consumption, Ding *et al.* [31] divided task scheduling into two stages. The first stage used a centralized task dispatcher based on the M/M/S queuing model to assign tasks to the host. According to the user's service level agreeement (SLA) requirements, the Q-learning-based scheduler maps tasks to VMs in the second stage, thereby minimizing energy consumption and task response time. In order to achieve the trade-off between profit and energy consumption, Kumar *et al.* [25] proposed an improved particle swarm optimization algorithm (PSO-COGENT), which can balance the execution time, cost, and energy consumption.

In the area of edge or fog, computation offloading is facing more challenges and diversities such as limited energy power and ephemeral resources [32]. Sun et al. [33] proposed a general IoT-fog-cloud architecture that fully exploits the advantages of fog and cloud and then presents energy and time-efficient computation offloading and resource allocation (ETCORA) algorithm to minimize energy and time cost. In [34], the authors formulated the computation offloading problem for mobile edge computing into the system cost minimization problem. They proposed a distributed algorithm consisting of offloading strategy by taking the completion time and energy into account.

2.2 Multi-objective task scheduling

In order to maximize the profit of cloud providers while satisfying user QoS, scheduling algorithms based on multi-objective optimization have been used to solve such two or more conflicting objectives. In [35], Abdullahi et al. proposed a discrete version of the symbiotic organism search algorithm to achieve optimal scheduling in terms of makespan, response time, and VM imbalance rate. In order to reduce the time-consuming problem of VM creation, Zhang et al. [36] proposed a two-stage task scheduling algorithm. Based on the historical task scheduling information, the authors used Bayesian classifiers to classify tasks and create corresponding VMs in advance. In the second stage, tasks are scheduled to be executed on the best matching VM type to minimize makespan and deadline violation rates.

Masadeh et al. [37] proposed a new metaheuristic called vocalization of humpback whale optimization algorithm (VWOA). They applied it to task scheduling problems in the cloud environment to reduce makespan, cost, energy consumption, and maximize resources utilization. Zhou et al. [22] propose a genetic algorithm with a greedy strategy to minimize the total execution time, response time, and cost of the task. Wu et al. [38] proposed an L-ACO and a heuristic algorithm ProLiS in workflow scheduling to minimize execution cost under deadline constrained. Sahoo et al. [39] used the learning automata (LA) scheduling algorithm to optimize the energy consumption and makespan for deadline-sensitive tasks. Although this method can greatly improve the task completion rate, it can not balance multiple objectives.

In [40], the sea lion optimization algorithm (SLnO) is applied to solve combinatorial optimization task scheduling problems in the cloud environment. The algorithm conserves more energy and increases resources utilization simultaneously compared with other meta-heuristic algorithms. PremJacob et al. [41] used a hybrid algorithm of cuckoo search and particle swarm optimization to reduce makespan, cost and improve completion rate. However, although minimizing makespan can indirectly improve task completion rate, it can not achieve the best performance. To solve the large-scale task scheduling optimization problem on Infrastructure as a Service (IaaS) cloud computing environment, Abdullahi et al. [42] proposed symbiotic organisms search algorithm with a chaotic optimization strategy to address multi-objective task scheduling.

The above survey result shows that meta-heuristic algorithms have been widely used to solve the multi-objective optimization scheduling problem in the cloud, which may benefit from the significant advantages in implementation, deployment, and performance. This article proposes an enhanced ant colony algorithm to solve the cloud's deadline-constrained multi-objective task scheduling problem. The goal of task scheduling in this paper is to improve task completion rate, minimize makespan and reduce energy consumption while achieving load balance. Furthermore, we use a modified backfilling algorithm to reorder the tasks in the VM waiting queue to further improve the completion rate of tasks. Unlike the current research work to improve the completion rate by minimizing the makespan, we take the completion rate as one of the optimization objectives in our scheduling algorithm. In [39], the authors regarded the completion rate as one of the optimization objectives in the scheduling algorithm, but it cannot achieve the balance of multiple objectives. The algorithms proposed in this paper can achieve the trade-off between completion rate, makespan, and energy consumption.

3 Task scheduling framework

A cloud data center consists of many heterogeneous servers, each hosting one or more VMs. VMs are heterogeneous, which have different efficient performances depending on the resources allocated to them [43]. The efficient task scheduling algorithm assigns tasks to appropriate VMs to make full use of cloud system resources and guarantee completion time.

In this section, we introduce the cloud task scheduling framework and discuss the related models as well as the optimization objective of the proposed scheduling algorithm. The main mathematical symbols used in this paper are listed in Table 1. The proposed scheduling framework, which is shown in Fig. 1, is designed for deadline-constrained multi-objective task scheduling in the cloud environment. The scheduling framework mainly consists of two phases. The first phase uses an enhanced ant colony algorithm to assign tasks to suitable VMs to minimize the makespan, improve task completion rate and reduce energy consumption. In the second phase, we use a modified backfilling algorithm to reorder tasks in the VM waiting queue, which can further improve the task completion rate. Next, we will introduce the model used in this paper and the objective function of task scheduling.

3.1 VM model and task model

In this paper, our scheduling objective is to find a mapping scheme between tasks and VMs to minimize makespan, energy consumption and improve task completion rate. Let $V = \{v_1, v_2, ..., v_n\}$ represents N VMs that are used to process tasks submitted by users, each v_j in V represents VM ID and with different computing capacity C_j (in terms of Million Instruction Per Second (MIPS)). $T = \{t_1, t_2, ..., t_m\}$ represents M tasks, $M > N$, each t_i in T represents task ID and consist of three attribute $t_i = <a_i, l_i, d_i>$, where a_i, l_i, d_i are the arrival time, task length (in terms of Million Instrution (MI)) and deadline of t_i. The final scheduling scheme can represent by a $M * N$ matrix S as following:

$$S = \begin{bmatrix} s_{11} & s_{12} & \dots & s_{1n} \\ s_{21} & s_{22} & \dots & s_{2n} \\ \dots & \dots & \dots & \dots \\ s_{m1} & s_{m2} & \dots & s_{mn} \end{bmatrix},$$

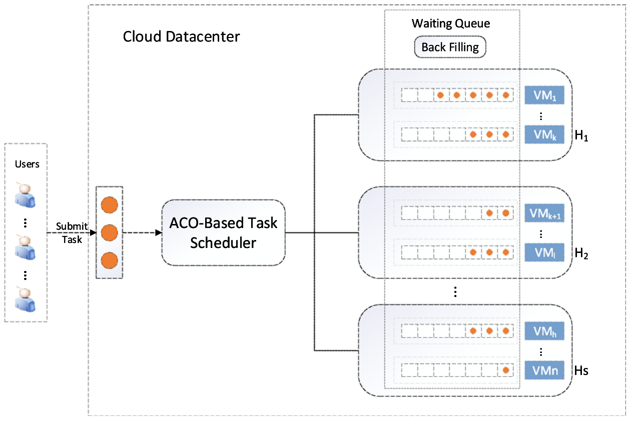


Fig. 1 EACO–MBF scheduling framework

where $s_{i,j}$ is a decision values, when $s_{i,j} = 1$, it represents task t_i allocated to VM v_j, otherwise $s_{i,j} = 0$. And for each task $i \in \{1, 2, ..., m\}$, $\sum_{j=1}^{N} s_{i,j} = 1$.

We model the estimated execution time of the task on the VM as the length of the task divided by the allocated VM's computing capacity. Therefore, the estimated execution time of the tasks on VMs can be represented by an M*N matrix ETC. Each row in the matrix ETC contains the estimated execution time of a task on each VM.

$$ETC = \begin{bmatrix} etc_1^1 & etc_1^2 & \dots & etc_1^n \\ etc_2^1 & etc_2^2 & \dots & etc_2^n \\ \dots & \dots & \dots & \dots \\ etc_m^1 & etc_m^2 & \dots & etc_m^n \end{bmatrix},$$

where etc_i^j represents the estimated execution time of task t_i on VM v_j, and it can calculated as follows:

Table 1 Main symbol used in this paper

Symbol	Description	Symbol	Description
t_i	Task ID of ith task	E_{a_j}	Energy consumption of v_j in active state
v_j	VM ID of jth VM	E_{idl_j}	Energy consumption of v_j in idle state
C_j	Computing capability of v_j (in terms of MIPS)	E_{total}	Total energy consumption in cloud
a_i	Arrival time of t_i	F	Objective function
l_i	Task length of t_i	τ_0	Initial pheromone value
d_i	Deadline of t_i	$\tau_{i,j}$	Pheromone value of combination (t_i, v_j)
$s_{i,j}$	Decision values indicate whether t_i allocate to v_j	$\eta_{i,j}$	Heuristic information of combination (t_i, v_j)
etc_i^j	Estimation execution time of t_i on v_j	E_i^j	Energy consumption of t_i execute on v_j
st_i^j	Start time of t_i on v_j	p_i^j	The probability of t_i allocate to v_j
ft_i^j	Finished time of t_i on v_j	CR	Completion rate
dc_i^j	Decision value indicate whether t_i execute on v_j can satisfy deadline.	$\bar{U}$	Average CPU utilization rate
E_j	Total Energy consumption of v_j	U_j	CPU utilization rate of v_j
μ_j	Total execution time of tasks assigned to v_j	U_{std}	Standard deviation of CPU utilizaiotn
Φ	Makespan		

$$etc_i^j = \frac{l_i}{C_j}. \tag{1}$$

The task's start execution time can be determined by the task's finish time in the tail of the VM waiting queue and the task's arrival time. Let st_i^j and ft_i^j respectively represent the start time and finish time of task t_i on VM v_j, so st_i^j can be calculated as follows:

$$st_i^j = max(ft_p^j, a_i). \tag{2}$$

This formula shows that task t_i can start its execution at latter of its own arrival time and the execution end time of the previous task. The finish time of the task t_i can be defined as:

$$ft_i^j = st_i^j + etc_i^j. \tag{3}$$

By comparing the finished time and the task's deadline, we can determine whether a task violates its deadline. Let binary variable dc_i^j indicates whether task t_i execution on the VM v_j can meet the deadline requirement, if met, $dc_i^j = 1$, otherwise $dc_i^j = 0$. The mathematical formula is as follows:

$$dc_i^j = \begin{cases} 1, & ft_i^j \le d_i \\ 0, & ft_i^j > d_i \end{cases}. \tag{4}$$

3.2 Energy model

In this paper, we mainly focus on the energy consumption of IT equipments in the cloud environment and establish an energy consumption model, like [39]. VMs in an idle state also consume plenty of energy, accounting for about 60%-70% of VMs in the active state. [9, 26, 44]. Let E_{a_j} represents the energy consumption of VM v_j in an active state, and E_{idl_j} represents the energy consumption of the idle state. E_j is the total energy consumption of the jth VM which considers the energy consumption of active state and idle state, calculated as follows:

$$E_j = (E_{a_j} + E_{idl_j}) \times C_j[39]. \tag{5}$$

The totla execution time μ_j of VM v_j can be calculated as:

$$\mu_j = \sum_{i=1}^{M} (s_{i,j} \times etc_i^j). \tag{6}$$

Makespan Φ is the maximum total execution time among all VMs, it can be expressed as:

$$\Phi = max(\mu_j), j \in \{1, 2, ..., N\}. \tag{7}$$

The energy consumption of VM v_j in active state and idle state are calculated as follows:

$$E_{a_j} = \mu_j \times \sigma_j, \tag{8}$$

$$E_{idl_j} = (\Phi - \mu_j) \times 0.6 \times \sigma_j[39], \tag{9}$$

where $\sigma_j = 10^{-8} \times (C_j)^2$ Joules/MI [26]. The total energy consumption of cloud system is denoted by E_{total}, and it can expressed as follows:

$$E_{total} = \sum_{j=1}^{N} E_j. \tag{10}$$

3.3 Objective function

This paper aims to minimize makespan and energy consumption while improving task completion rate. When multiple indicators conflict in multi-objective optimization, it becomes more challenging. To improve task completion rate, tasks will be scheduled to execute on a high-performance VM as much as possible, which will cause more energy consumption and lead to load imbalance, and vice versa.

In this paper, we construct a multi-objective function that considers the makespan and energy consumption. Based on the characteristics of the ant colony algorithm using pheromone optimization, the deadline constraints are introduced in the pheromone update rule, which only considers the objective function in the standard ACO. In this way, we can achieve the best performance between makespan, energy consumption, and completion rate. The objective function is as follows:

$$F = min\left(\omega \times \frac{\Phi}{\Phi_{max}} + (1-\omega) \times \frac{E_{total}}{E_{max}}\right), \tag{11}$$

where Φ_{max} is the maximum allowable makespan, E_{max} represents the maximum energy consumption that the cloud system may generate. ω $(0 \le \omega \le 1)$ is a weight coefficient that can be adjusted in practice. For example, to meet the user's QoS and improve task's response time, we can set $\omega > 0.5$, whereas in an energy-aware environment, we can set $\omega < 0.5$. When ω is equal to 0 or 1, the scheduling problem will be more straightforward because multi-objective optimization scheduling is simplified to a single-objective optimization scheduling.

4 EACO–MBF scheduling model

This section describes the detailed implementation of our proposed EACO–MBF algorithm that attempts to minimize the performance metrics (i.e., makespan, energy consumption) and satisfy task deadline constraints.

4.1 Enhanced ant colony optimization algorithm

Ant colony algorithm was firstly proposed by [45] to solve the traveling salesman problem. The basic idea of the ant colony algorithm is that in each iteration of the optimization process, the ants gradually gather on the optimal path according to the pheromone and heuristic information. Finally, all ants concentrate on the best path under positive feedback.

This paper uses the ant colony algorithm to solve the deadline-constrained multi-objective task optimization scheduling problem. Based on the characteristics of the ant colony algorithm according to pheromone optimization, we add the deadline of the task to the pheromone update rule to achieve the best performance in makespan, energy consumption, and task completion rate. In the enhanced ant colony algorithm, the pheromone update rule of the ant colony algorithm is modified. The evaporation and accumulation of pheromone are based not only on the fitness function but also on whether the task can meet the deadline requirement on the assigned VM. Using the Enhanced ACO, we can better trade off multiple objects, such as completion rate, makespan, and energy consumption.

Algorithm 1 Pseudo code of EACO Algorithm

```
Input: T, V, $\alpha$, $\beta$, antNum, itNum, q0, N, M, S.
Output: $S^*$.
 1: Initial all element in Matrix S to 0;
 2: Generate ETC matrix using Equation 1;
 3: Generate pheMatrix using Equation 12;
 4: Generate Heuristic Matrix using Equation 13;
 5: $k \leftarrow 0$;
 6: $bestFitness \leftarrow 0$;
 7: while $k < itNum$ do
 8:     for i=1 to antNum do
 9:         for j=1 to M do
10:             Generate a random variable $q \in [0,1]$;
11:             if $q \le q_0$ then
12:                 $v \leftarrow$Choose a destination VM using Equation 14;
13:             else
14:                 $v \leftarrow$Choose a destination VM using Equation 15;
15:             end if
16:             $s_{j,v} = 1$;
17:             Update local pheromone using Equation 17;
18:         end for
19:         currFitness ← Calculate fitness value using Equation 11;
20:         if currFitness ¿ bestFitness then
21:             bestFitness ← currFitness;
22:             $S^* \leftarrow S$;
23:         end if
24:         Update global pheromone using Equation 19;
25:     end for
26: end while
```

4.1.1 Definition of pheromone and heuristic information

In the real world, the pheromone is a chemical for ants to communicate with each other. Ants find food by sensing the pheromone left by other ants [45]. In this paper, pheromone was represented by the combination of task and VM. The more the accumulated pheromone, the more ants select this VM in prior iteration and prove that the mapping scheme is better. Let $\tau_{i,j}$ represent the pheromone value of combination (t_i, v_j). The initial pheromone value is very important because it greatly influences the optimization efficiency of the ACO. We set the initial pheromone value τ_0 as follows:

$$\tau_0 = \frac{1}{N}. \tag{12}$$

Apart from pheromone, heuristic information is another important factor in the ant colony algorithm. Heuristic information $\eta_{i,j}$ represent the expected value that allocate task t_i to VM v_j. In this paper, heuristic information $\eta_{i,j}$ is represented by the combination of estimated execution time and energy consumption required for task t_i execute on VM v_j. The larger the $\eta_{i,j}$, the shorter execution time and less energy consumption can be obtained by assigning task t_i to VM v_j. We set $\eta_{i,j}$ to a small enough number when task t_i execute on VM v_j can not satisfy it's deadline (10^{-3} in this paper), so that in the process of optimization, the probability of task t_i selecting VM v_j is close to 0. In this way, we can reduce the search space of ants in the optimization process, improve the algorithm's convergence speed, and find the optimal solution in a shorter time. The mathematical expression of $\eta_{i,j}$ is as follows:

$$\eta_{i,j} = \begin{cases} \dfrac{1}{etc_i^j \times E_i^j} & ft_i^j \le d_i \\ 10^{-3} & ft_i^j > d_i \end{cases}, \tag{13}$$

where E_i^j represent the energy consumption of task t_i execute on VM v_j.

4.1.2 Pseudo-random-proportional and pheromone updating rules

In each iteration of the ant colony algorithm, the ant will select the next target based on the value of pheromone and heuristic information and determine which VM to choose next according to a probabilistic behavior selection rule called pseudo-random ratio. The expression is as follows:

$$dj = \begin{cases} \underset{J}{\arg\max}[\tau_{i,j}]^{\alpha} \cdot [\eta_{i,j}]^{\beta} & q \le q_0 \\ J & q > q_0 \end{cases}, \tag{14}$$

where q_0 is a value between [0, 1], q is a parameter uniformly distributed between [0, 1], α and β are weighting factors that determine the degree of importance between pheromone and heuristic, and J is a random variable. The value of $[\tau_{i,j}]^{\alpha} \cdot [\eta_{i,j}]^{\beta}$ represents the expected value of selecting VM v_j for task t_i, it was jointly determined by pheromone $\tau_{i,j}$ and heuristic information $\eta_{i,j}$. The formula 14 indicates that the probability of the ant chooses the optimal task and VM mapping scheme is q_0. At this time, the ant continues to develop the existing knowledge. Moreover, explore other possible scheduling schemes with the probability of $1 - q_0$ to avoid falling into the local optimal.

In this paper, the random variable J selects the target VM using the roulette way. The formula for calculating the probability of roulette is:

$$p_{i,j} = \frac{[\tau_{i,j}]^{\alpha} \cdot [\eta_{i,j}]^{\beta}}{\sum_{v_j \in V} [\tau_{i,j}]^{\alpha} \cdot [\eta_{i,j}]^{\beta}}, \tag{15}$$

$p_{i,j}$ represents the probability of task t_i choosing VM v_j, α and β are weight coefficients, which determine the influence of pheromone and heuristic information, respectively. We use the pseudo-random proportional rule to select the target VM for each task. When $q \le q_0$, the task will select the VM with the product of the largest pheromone and heuristic information. Otherwise, it will be based on roulette to explore other possible mapping schemes.

After each task selects the destination VM, the local pheromone will be updated. This process evaporates part of the pheromone so that the probability of other ants selecting the same VM will reduce, and the chance of other unselected VMs is increased. The local pheromone update rule is as follows:

$$\tau_{i,j} = (1 - \xi) \times \tau_{i,j} + \xi \times \tau_0, \tag{16}$$

where ξ is the pheromone evaporation coefficient, satisfying $0 < \xi < 1$, which indicates the degree of pheromone evaporation, and τ_0 is the value of the initial pheromone. The smaller the ξ, the more pheromone will be evaporated. Otherwise, more pheromones will be left.

In this paper, our scheduling objective is to minimize the makespan, energy consumption, and improve task completion rate. We can only achieve minimize makespan and energy consumption if we use the standard ACO. Although minimizing the makespan can improve task completion rate, it cannot achieve the best performance. Inspired by [39], we introduced task deadlines into the pheromone update rules to improve the task completion rate. The improved local pheromone update rules are as follows:

$$\tau_{i,j} = \begin{cases} (1 - \xi) \times \tau_{i,j} + \xi \times \tau_0, & ft_i^j \le d_i \\ \tau_{i,j} \times \phi & ft_i^j > d_i \end{cases}, \tag{17}$$

where ϕ is the penalty coefficient, and $0 < \phi < 1$ means to punish the pheromone combination (t_i, v_j) that cannot meet the deadline. The pheromone will be reduced to a minimum so that other ants have the slightest chance of choosing this VM. When all the ants in the population complete an iterative optimization process, we will select the optimal ant according to the formula 11. If the ant is the optimal ant so far, then it is regarded as the optimal ant. Otherwise, it was the optimal ant before this iteration. Let S^* represent the optimal scheduling scheme generated by the optimal ant in this iteration, and update the global pheromone according to this optimal scheme as follows:

$$\tau_{i,j} = (1 - \xi) \times \tau_{i,j} + \xi \times F(S^*). \tag{18}$$

Algorithm 2 Pseudo code of Modified Back Filling

```
Input: $t_i$,$v_j$.
Output: waitingList. /*waiting List of VM j*/.
1: waitingList ← $v_j.waitingList()$;
2: len ← waitingList.size();
3: if len == 0  $t_i$ is satisfy deadline then
4:     insert $t_i$ to waitingList;
5: else
6:     while $t_i$ not reach list head do
7:         $t_j$ ← Get previous task of $t_i$ in waiting list;
8:         if $t_j$ is satisfy deadline then
9:             if insert $t_i$ to the previous of $t_j$ and $t_j$ also satisfy deadline then
10:                insert $t_i$ to the previous of $t_j$;
11:                if $t_i$ is satisfy deadline then
12:                    break;
13:                end if
14:            else
15:                break;
16:            end if
17:        else
18:            insert $t_i$ to the previous of $t_j$;
19:            if $t_i$ is satisfy deadline then
20:                break;
21:            end if
22:        end if
23:    end while
24: end if
```

The global pheromone update can increase the pheromone of each combination in the optimal scheduling scheme S^* to increase the chances of ants selecting these combinations in subsequent iterations, and the ants will gradually aggregate in the optimal path. In the local pheromone update rule, we introduce the task deadline condition, thereby reducing VMs that cannot meet the selected task deadline. Similarly, in the global pheromone update rules, we introduce task deadline conditions to modify the global pheromone update rules. The revised global pheromone update rules are as follows:

$$\tau_{i,j} = \begin{cases} (1 - \xi) \times \tau_{i,j} + \xi \times F(S^*) & ft_i^j \le d_i \\ (1 - \xi) \times \tau_{i,j} & ft_i^j > d_i \end{cases}, \tag{19}$$

the new update rule Eq. (19) indicates that only the combinations in the current optimal scheduling schema S^* that can meet the task deadline are allowed to release pheromone. Otherwise, these combinations only vaporize pheromone. This can increase the selection chance of other VMs that can meet the task deadline. The process of the EACO is shown in Algorithm 1.

4.2 Modified backfilling algorithm

Backfilling algorithms are widely used in cloud computing to improve resource utilization [4]. The backfilling algorithm finds time slots in computing resources and selects appropriate tasks to execute in these time slots, thereby improving resource utilization and task completion rate. In this paper, based on the backfilling algorithm's characteristics, we use a modified backfilling algorithm to reorder the tasks in the VM waiting queue so that more tasks can meet the deadline. The more implementation details of MBF will be discussed in the following example.

As shown in Fig 2a, at the current moment, VM1 and VM2 are executing two tasks, respectively. At times t=10 and t=11, task T5 and task T6 arrive at the cloud data center and are scheduled by the EACO scheduler to VM1 for execution. The execution time and deadline of task T5 are 4 and 18, respectively, while T6 is 2 and 15. Since T3 has not been executed yet, both tasks need to wait in the waiting queue to be scheduled for execution. If the tasks in the waiting queue are executed according to the first-come-first-served(FCFS) scheduling algorithm, as shown in Fig. 2b, only T5 can meet the deadline requirement, and T6 cannot. Since task T5 and task T6 have arrived in VM when task T3 finishes executing, we use the MBF algorithm for each task that arrives in VM (as shown in Algorithm 2). As shown in Fig. 2c, T5 will directly insert into the queue because task T3 has not been executed and the VM waiting queue is empty (lines 2-3). When T6 arrives, because the waiting queue is not empty, task T6 calculates its finish time and checks whether it can satisfy the deadline. Since the deadline is not met, the backfilling strategy is executed as follows: backfilling task T6 before task T5, if the deadline is met after backfilling, and task T5 is not broken, or task T5 does not meet the deadline in previous, the backfilling is completed. Otherwise, the backfilling strategy is not executed (lines 6–19). As shown in Fig. 2c, by performing the MBF algorithm, both tasks T5 and T6 can meet the deadline.

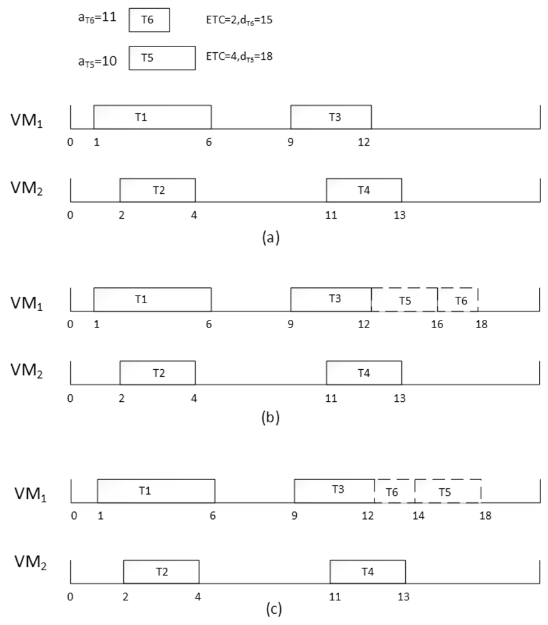


Fig. 2 Tasks await scheduling (**a**) and execution orders without MBF (**b**) and with MBF (**c**)

4.3 Complexity analysis

In this section, the computational complexity of the proposed EACO–MBF algorithm is analyzed. The complexity of the EACO–MBF is depended on the complication of the EACO and MBF. Consequently, the complexity of the EACO–MBF is given as follows:

$O(EACO--MBF) = O(EACO) + O(MBF)$ where, $O(EACO) = O(nI \times nA \times M \times N)$, $O(MBF) = O(M^2)$, where nI denotes the iteration count, nA means the ant count, M indicates the number of tasks, and N signifies the number of VMs. The MBF algorithm is the variants of the insertion sort algorithm, so its complexity is $O(M^2)$.

5 Experiment setup and performance evaluation

In this section, the experiment environment, evaluation metrics, performance evaluation, and results analysis are reported.

5.1 Experiment environment

The effectiveness of the proposed task scheduling method using EACO–MBF is evaluated and compared with four state-of-the-art published methods in this section. This includes (i) *LA* [39], (ii) *DSOS* [35], (iii) *IWC* [18], and (iv) standard *ACO*. We implemented the proposed method and other compared algorithms on the cloud computing simulation toolkit Cloudsim, which is a scalable cloud simulation platform [46] and widely used in the cloud computing research [36, 47].

In our simulation experiment, one data center was created and containing 48 hosts. Half of the hosts are HP

ProLiant ML110 G4, and the other half is HP ProLiant ML110 G5. Four kinds of VM type is created in this simulation environment. The specific parameters of Hosts and VMs are listed in Table 2 and Table 3. The main parameters used for each algorithm were presented in Table 4. The parameters of IWC and LA are set according to [18, 39], and the other parameters of EACO are the same as ACO. We set the search agents (whales, ants) in the IWC, DSOS, ACO, and EACO algorithms to 100, and the number of iterations is 100. We repeat each experiment 20 times and use the average value as our final experimental result to avoid the influence of uncertain factors.

5.2 Evaluation metrics

Four performance metrics: completion rate, makespan, total energy consumption, and imbalance rate are used to evaluate the effectiveness of the proposed algorithm. We have discussed the calculation of makespan and total energy consumption in Sect. 3.

Task completion rate CR is the ratio between the number of tasks that meet their deadline and the total number of tasks. The mathematical expression is as follows:

$$CR = \frac{M_{succ}}{M}, \tag{20}$$

where M_{succ} represents the number of tasks that can meet the deadline, and M is the total number of tasks.

Degree of imbalance reflect whether the task scheduling algorithm can fully use all computing resources. The smaller the imbalance rate, the best the performance algorithm.

$$U_{std} = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(U_j - \bar{U})^2}. \tag{21}$$

In this paper, we evaluate the algorithm's performance on imbalance metrics by calculating the standard deviation U_{std} of the CPU utilization of the virtual machine. As shown in Eq. (21). Average CPU utilization rate $\bar{U}$ of VM is calculated as follows:

$$\bar{U} = \frac{\sum_{j=1}^{N} U_j}{N}, \tag{22}$$

$$U_j = \frac{\sum_{i=1}^{M}(s_{i,j} \times etc_i^j)}{FT}, \tag{23}$$

where U_j represents the CPU utilization rate of VM v_j, which can be calculated by the Eq. (23), and FT represents the finish time of the final task.

SLA reflect the preference of user service requirements. We calculate the average SLA violation (ASLAV) to evaluate the performance of the proposed algorithm and other comparison methods. The formula definition of ASLAV can be expressed as the following formula:

$$ASLAV = \frac{1}{K}\sum_{i=1}^{K}\frac{ft_i - d_i}{d_i - a_i}, \tag{24}$$

Table 4 Main parameters used in experiment for each algorithm

Algorithm	Parameter	Value	Description
IWC	b	1	Spiral searching path
	PSmax	51	The largest population
	PSmin	10	Initial population
	α	0.25	Individual generate
	γ	20	Nonlinear factor
LA	Φ	0.1	Reward coefficient
	ϕ	0.1	Penalty coefficient
ACO	α	1	Weighting factor
	β	1	Weighting factor
	ξ	0.1	Evaporation coefficient
	q_0	0.5	Probability parameter
EACO	ϕ	0.1	Penalty coefficient

Table 2 Host configuration used in simulation

Host type	CPU (MIPS)	Core	RAM (GB)	BW (Gb/s)
HP ProLiant ML110 G4	1860	2	4	1
HP ProLiant ML110 G5	2660	2	4	1

Table 3 Four kinds of VM types

VM type	CPU (MIPS)	RAM (GB)	BW (Mb/s)
High-CPU Medium Instance	2500	0.85	100
Extra Large Instance	2000	3.75	100
Small Instance	1000	1.7	100
Micro Instance	500	0.613	100

where K denotes the number of tasks that can not meet the deadline, $ft_i - d_i$ indicates the delay time that the task exceeds the deadline. $d_i - a_i$ is the maximum approval time to be executed.

5.3 Performance evaluation based on synthetic datasets

In this scenario, a workload generator was developed to generate deadline-constrained tasks and submit them to the cloud datacenter. The arrival of the task is model using Poisson distribution where inter-arrival time is exponential distribution [48], and the following formula generates the deadline of the task:

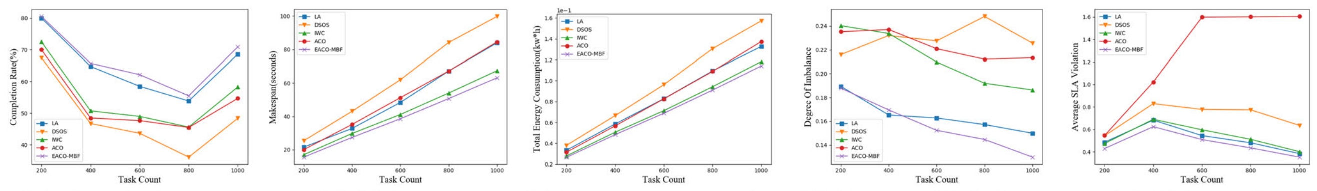
$$d_i = a_i + baseD, \tag{25}$$

where *baseD* is a random variable uniformly distributed in (2, 5), task size generate from 3000 to 10000 MI (Millons Instruction), and the arrival rate of the task is 40. Vary the task count in the range [200, 1000] in the interval of 200 to evaluate each algorithm's performance.

Figure 3a presents the task completion rate of all algorithms for executing different task counts (200–1000). The figure shows that our algorithm obtains the best performance in all task cases. LA achieve the second-best completion rate, while other algorithms competitive together. DSOS, IWC, and ACO try to minimize the makespan, which indirectly improves the task completion rate. However, the experimental results show that this method cannot achieve the best performance. EACO–MBF and LA introduce the deadline condition in the pheromone update rule and reinforcement signal, respectively, to obtain better performance. Moreover, the MBF strategy tuning the task order in the VM waiting queue further improves our algorithm's completion rate.

Figure 3b illustrates the makespan value of all algorithms. The figure indicates a minimization of makespan using EACO–MBF in all task instances. The percentage improvement of EACO–MBF over LA in makespan summarize in Table 5. The tables show that the makespan values of EACO–MBF compared with LA are decrease 26.61%, 23.94%, 23.45%, 23.51%, and 20.00% on average in task count 200, 400, 600, 800, and 1000 respectively. Additionally, EACO–MBF acquired the best performance improvement at task 200, reducing 26.61% in makespan values. Our algorithm obtains the minimum makespan due to the initialization method of heuristic information. Those VMs that cannot meet the task deadline will have a longer execution time. The heuristic information of these VMs set to a small enough value in the initial state to reduce the probability of selecting these VMs in the optimization process.

The experimental results of energy consumption are shown in Fig. 3c. It indicates that EACO–MBF acquires the minimization of energy consumption compared with other scheduling strategies, followed by IWC. The percentage improvement of EACO–MBF over LA in energy consumption summarize in Table 6. The tables show that



(a) Completion rate (b) Makespan (c) Energy consumption (d) Degree of imbalance (e) SLA metric

Fig. 3 Performance evaluation based on synthetic datasets

Table 5 Comparison of makespan obtained by LA and EACO–MBF for synthesis datasets

Task size	LA			EACO–MBF			Improvement			t-test calculate t-value
	Best	Worst	Avg	Best	Worst	Avg	Best(%)	Worst(%)	Avg(%)	
200	17.21	26.58	21.24	15.07	16.30	15.59	12.40	38.66	26.61	9.68
400	30.73	46.78	36.03	26.24	28.35	27.40	14.60	39.39	23.94	8.57
600	44.18	58.80	50.41	37.61	39.30	38.59	14.87	33.17	23.45	12.73
800	56.95	81.20	66.26	49.35	51.75	50.68	13.34	36.26	23.51	11.83
1000	68.79	95.98	78.87	61.49	64.26	63.10	10.61	33.05	20.00	9.45

Table 6 Comparison of energy consumption obtained by LA and EACO–MBF for synthesis datasets

Task size	LA			EACO–MBF			Improvement			t-test calculate t-value
	Best	Worst	Avg	Best	Worst	Avg	Best(%)	Worst(%)	Avg(%)	
200	0.03	0.04	0.03	0.03	0.03	0.03	9.19	31.46	20.37	9.69
400	0.05	0.07	0.06	0.05	0.05	0.05	10.13	30.65	17.52	8.56
600	0.08	0.09	0.08	0.07	0.07	0.07	10.06	24.81	16.85	12.55
800	0.10	0.13	0.11	0.09	0.09	0.09	9.17	27.41	16.79	11.82
1000	0.12	0.15	0.13	0.11	0.12	0.11	6.85	24.69	14.05	9.47

the energy consumption of EACO–MBF compared with LA is reduced 20.37%, 17.52%, 16.85%, 16.79%, and 14.05% on average in task count 200, 400, 600, 800, and 1000 respectively. Moreover, EACO–MBF acquired the best performance improvement at task 200, reducing 20.37% in energy consumption. This part shows that our algorithm can save more energy while executing the task.

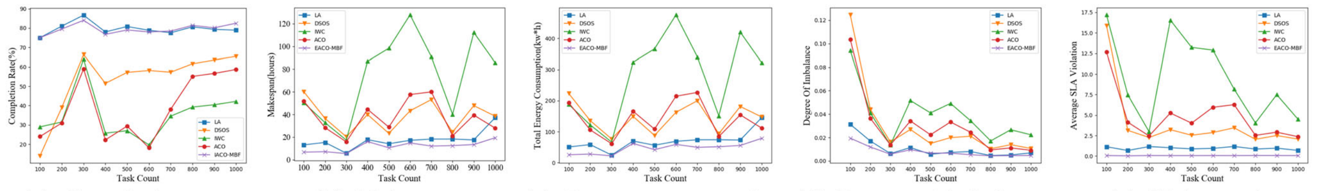
The degree of imbalance of EACO–MBF performance compared with other well-knowing algorithms are present in Fig. 3d. DSOS, IWC, and ACO obtain a terrible balance rate because these methods minimize makespan to achieve load balance. However, it can not get the minimal value because the fitness function contains several performance metrics in the multi-objective optimization problem. EACO–MBF and LA have deadline conditions in their pheromone update rule and reinforcement signal, respectively. The increased makespan will cause more tasks can not complete before time constraints. Therefore, these two methods try to search for other solutions. LA is a deadline-sensitive algorithm. It will not choose a solution with a smaller makespan but cannot meet the time constraints of the tasks. The figure shows that our algorithm obtains the best performance among most task cases. It means that our scheduling method can evenly allocate the task among all computing resources to improve the resource utilization rate.

Figure 6e shows the average SLA violation rate of all algorithms. We can observe that EACO–MBF obtained the minimal SLA violation rate compared to others methods. EACO–MBF tries its best to find a solution that can minimize the execution time of the tasks. Although some tasks can not meet deadline requirements, they get the slightest time delay to complete the task. LA, IWC, and DSOS have moderate optimization effects, while ACO has the worst performance. It is the ACO obstacle that easily traps into a local-optimal solution. This part shows that our algorithm can achieve a better trade-off between energy consumption and SLA.

5.4 Performance evaluation based on real trace datasets

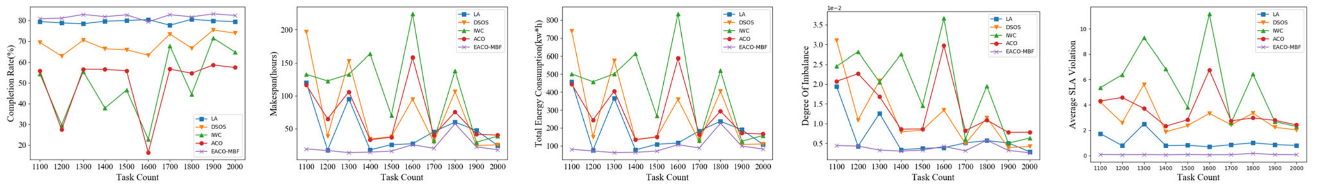
The most reliable approach to evaluate task scheduling algorithms is to use real datasets obtained from the large-scale distributed system. In this part of the experiment, the tasks obtained from google trace [49] and planetlab [50] datasets are used to access the proposed method in Cloudsim. Google trace contains 25 million tasks grouped in 650 thousand jobs that span 29 days. It is not easy to experiment on google trace for all tasks because the task count was too much. In this paper, we adopt workload generated based on [51], which presents a comprehensive analysis of the workload characteristics derived from Google datacenter that features approximately 25 million tasks. Task length and CPU utilization rate generate from Lognormal and Weibull distribution. In this scenario, to generate a deadline for each task, we utilize the approach adopted in [47]. First, a baseline duration is defined for each task. It is calculated as the ratio of task length to the average capacity of VMs within the data center. Completion deadline is defined as $(1+\chi) \times \textit{baseline duration}$, where χ is termed stringency factor, representing a degree of difficulty in meeting the deadlines. A larger χ indicates a more relaxed deadline. We set χ as 1.2 in our experiment. PlanetLab is from the common system, which monitors the operation of the PlanetLab infrastructure and collects data from each node of the PlanetLab [50]. To prove the effectiveness of our algorithm, we conduct experiments on the data 20110303, 20110306, 20110309, 20110322, and 20110325.

Figures 4a, 5a and 6a present all algorithms' task completion rates using the google trace dataset and PlanetLab dataset. Figure 4a shows that in small task instances (100-600), the completion rate of EACO–MBF is slightly lower than LA; when task instances are more than 600, our algorithm gets the better performance. This can be confirm by Fig. 5a. The figure shows that EACO–MBF performs better than other algorithms in large task instances (1000–2000). While in Fig. 6a, the proposed algorithm EACO–MBF is slightly lower than LA. The reason is that



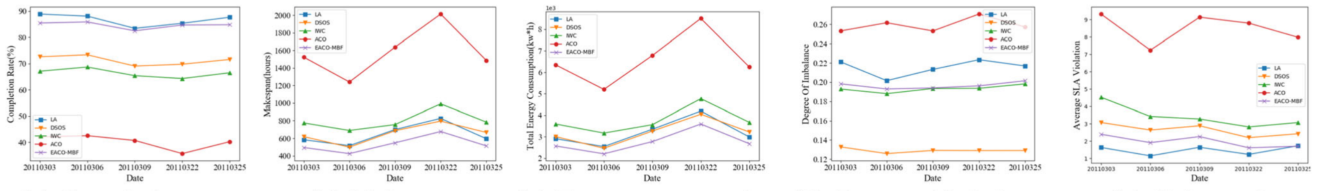
(a) Completion rate (b) Makespan (c) Energy consumption (d) Degree of imbalance (e) SLA metric

Fig. 4 Performance evaluation for executing small tasks on google trace datasets



(a) Completion rate (b) Makespan (c) Energy consumption (d) Degree of imbalance (e) SLA metric

Fig. 5 Performance evaluation for executing large tasks on google trace datasets



(a) Completion rate (b) Makespan (c) Energy consumption (d) Degree of imbalance (e) SLA metric

Fig. 6 Performance evaluation based on planetlab datasets

Table 7 Comparison of makespan obtained by LA and EACO–MBF for google trace datasets

Task size	LA			EACO–MBF			Improvement			t-test calculate t-value
	Best	Worst	Avg	Best	Worst	Avg	Best(%)	Worst(%)	Avg(%)	
200	15.26	15.27	15.26	5.53	9.43	7.10	63.78	38.22	53.51	29.56
400	17.58	17.77	17.72	9.59	20.42	16.00	45.47	− 14.93	9.70	2.969
600	17.14	17.15	17.14	12.05	20.94	14.78	29.66	− 22.15	13.75	4.244
800	18.27	18.34	18.28	8.82	16.29	12.49	51.72	11.16	31.68	12.221
1000	37.11	37.13	37.12	9.77	29.07	19.38	73.66	21.69	47.79	17.281
1200	17.55	17.57	17.56	10.71	21.74	16.87	38.95	− 23.75	3.93	1.022
1400	17.86	18.76	17.96	9.06	22.77	14.49	49.25	− 21.37	19.32	5.13
1600	25.71	29.30	27.14	15.49	32.68	25.69	39.77	− 11.55	5.35	1.422
1800	58.95	64.41	59.99	26.81	76.11	57.55	54.51	− 18.17	4.07	0.69
2000	24.67	24.77	24.68	13.21	28.88	18.05	46.44	− 16.62	26.85	8.811

all tasks have the same length in the PlanetLab dataset, and LA expenses more energy to guarantee the task completion rate.

Figures 4b, 5b and 6b show the makespan value obtains by all task scheduling algorithms. The figures show that the proposed EACO–MBF algorithm also reached the minimal makespan value using two real datasets. The percentage improvement of EACO–MBF over LA in makespan is summarized in Tables 7 and 8. From Tables 7 and 8, we can see that our algorithm has excellent performance improvement compared to LA. Especially at task 200 in Google Trace and at date 20110309 in PlanetLab, we get the best improvement, 53.51% and 21.60% in makespan values, respectively.

Table 8 Comparison of makespan obtained by LA and EACO–MBF for planetlab datasets

Date	LA			EACO–MBF			Improvement			t-test calculate t-value
	Best	Worst	Avg	Best	Worst	Avg	Best (%)	Worst (%)	Avg (%)	
20110303	498.17	747.16	583.17	435.28	645.30	494.00	12.62	13.63	15.29	5.103
20110306	421.65	595.31	515.36	357.32	579.46	424.80	15.26	2.66	17.57	6.36
20110309	595.67	798.67	699.32	444.16	682.30	548.26	25.44	14.57	21.60	7.056
20110322	732.87	906.53	826.63	601.04	817.45	676.90	17.99	9.83	18.11	8.865
20110325	538.02	682.95	594.77	452.44	588.66	513.13	15.91	13.81	13.73	5.886

Table 9 Comparison of energy consumption obtained by LA and EACO–MBF for google trace datasets

Task size	LA			EACO–MBF			Improvement			t-test calculate t-value
	Best	Worst	Avg	Best	Worst	Avg	Best (%)	Worst (%)	Avg (%)	
200	58.38	58.54	58.46	21.93	36.36	27.79	62.44	37.89	52.46	29.861
400	69.36	70.18	69.92	38.97	78.40	62.18	43.82	− 11.71	11.08	3.666
600	69.02	69.25	69.13	48.69	81.21	59.05	29.44	− 17.27	14.58	4.968
800	74.03	74.39	74.17	38.19	65.26	51.61	48.42	12.28	30.42	12.941
1000	145.31	145.63	145.50	43.39	113.19	78.77	70.14	22.28	45.86	17.813
1200	75.45	76.01	75.71	49.95	88.50	71.18	33.80	− 16.43	5.99	1.871
1400	77.32	80.88	77.80	44.69	94.82	63.89	42.20	− 17.23	17.88	5.554
1600	110.49	123.97	115.83	69.61	132.70	106.24	37.00	− 7.04	8.29	2.618
1800	232.54	252.96	236.58	112.58	294.89	225.85	51.58	− 16.58	4.54	0.822
2000	107.50	108.07	107.68	65.36	121.32	82.25	39.21	− 12.26	23.61	9.376

The comparison results of energy consumption among the proposed EACO–MBF algorithm with other comparative algorithms are given in Figs. 4c and 5c for google trace datasets and Fig. 6c for PlanetLab datasets. It can be observed from these figures that the proposed algorithm achieved better energy saving compared with other methods. The percentage improvement of EACO–MBF over LA in energy consumption is summarized in Tables 9 and 10. From Tables 9 and 10, we can see that our algorithm obtained a significant performance improvement compared with LA. The best improvement is 52.46% at task instances 200 in google trace and 17.22% at date 20110309 in PlanetLab datasets.

Figures 4d, 5d and 6d show that EACO–MBF also acquires the best performance for imbalance rate in Google Trace; however, in the PlanetLab dataset, DSOS outperforms our algorithm. This is because the processing capacity of micro instance types of VMs was too weak, so these VMs can not satisfy the deadline requirement when executing PlanetLab tasks. In this way, the proposed algorithm EACO–MBF and LA do not schedule tasks to these VMs, which results in a high imbalance rate.

All algorithms' average SLA violation rate is presented in Figs. 4e, 5e, and 6e. In this scenario, we can see that LA outperforms EACO–MBF. The reason is the tasks in PlanetLab datasets have the equivalent task length, and the problem becomes how to allocate these tasks on different VM types. Our algorithm intends to trade-off energy consumption and SLA. Therefore, it needs to sacrifice a little SLA to balance the scales.

In order to show the improvement of our proposed algorithm, one side t-test was conducted to examine whether the makespan and energy consumption obtained by EACO–MBF is significantly less than LA for all task instances using the same stopping criteria. Acceptable errors of 1% are used to test the critical value obtained from the statistical table as 3.319. The statistical analysis of the performance of EACO–MBF and LA under different datasets are present in Tables 5-10. The tables show that the calculated t-value is greater than 3.319, which means a significant difference between the performance of EACO–MBF and LA for all datasets.

Table 10 Comparison of energy consumption obtained by LA and EACO–MBF for planetlab datasets

Task size	LA			EACO–MBF			Improvement			t-test calculate t-value
	Best	Worst	Avg	Best	Worst	Avg	Best (%)	Worst (%)	Avg (%)	
20110303	2590.49	3513.34	2906.41	2353.18	3131.39	2569.01	9.16	10.87	11.61	5.21
20110306	2210.19	2842.24	2539.73	1943.76	2780.29	2201.23	12.05	2.18	13.33	6.40
20110309	2961.89	3711.49	3347.63	2397.42	3255.36	2771.30	19.06	12.29	17.22	7.27
20110322	3854.57	4482.79	4191.33	3318.70	4107.81	3592.09	13.90	8.36	14.30	9.70
20110325	2780.26	3302.42	2988.93	2436.90	2961.35	2673.35	12.35	10.33	10.56	6.19

Table 11 Comparison of completion rate obtained by algorithm combined with MBF or not in synthesis datasets

Arrival rate	–					MBF					Improvement				
	LA	DSOS	IWC	ACO	EACO	LA	DSOS	IWC	ACO	EACO	LA (%)	DSO S(%)	IWC (%)	ACO (%)	EACO (%)
40	981	913	955	627	972	980	902	955	660	971	− 0.10	− 1.20	0.00	5.26	−0.10
60	377	278	299	301	353	394	294	317	316	370	4.51	5.76	6.02	4.98	4.82
80	437	330	369	311	429	462	335	401	322	446	5.72	1.52	8.67	3.54	3.96
100	188	157	161	197	173	221	184	187	215	209	17.55	17.20	16.15	9.14	20.81
120	128	117	109	157	120	171	148	147	181	162	33.59	26.50	34.86	15.29	35.00

5.5 Evaluate the effectiveness of MBF

To prove the effectiveness of our proposed algorithm MBF, we compare the completion rate of five algorithms LA, DSOS, IWC, ACO, and EACO, whether they combine MBF or not. In this experiment, we set the number of tasks to 1000 and the number of VMs to 60 and varied the arrival rate in the range [40,120] in the interval 20. The experimental results are shown in Table 11. Table 11 shows that the MBF algorithm can further increase the number of tasks that meet the deadline to improve the completion rate of tasks, especially as the arrival rate of tasks increases, the better the performance of the algorithm. This is because as the task arrival rate increases, the number of tasks arriving at the same time will also increase. As a result, some short tasks or tasks with tight deadlines are queued after long tasks or tasks with loose deadlines, which causes these tasks to fail to complete before the deadline, and our MBF algorithm can improve this situation.

6 Conclusion and future works

This article discusses the research work of task scheduling problems in the cloud environment. Furtherly introduce the challenge of task schedulers to find an energy-efficient solution for real-time tasks. We present a two-stage scheduling method for deadline-constrained tasks in cloud computing to tackle such an issue. The EACO and MBF are proposed to solve and optimize the task scheduling scheme by considering the makespan, completion time, and energy. The experimental results show that compared with other well-known task scheduling methods, our method can effectively reduce makespan by 25.28% and energy consumption by 23% on average. Finally, the results are analyzed using statistical t-tests, which show that EACO–MBF significantly improved the results.

The disadvantage of our work is the stability of the proposed algorithm EACO. It means that the experimental results of our method fluctuate significantly in repeated experiments. Sometimes it may obtain a relatively undesirable result. The reason is the algorithm is trapped into a local-optimal solution. In the future, we hope to combine the advantages of other heuristics and meta-heuristics to solve this problem, such as A* [52] and hybrid harmony algorithm [53].

Acknowledgements This work was supported in part by the Guangdong Major Project of Basic and Applied Basic Research under Grant 2019B030302002, in part by the Science and Technology Major Project of Guangzhou under number 202007030006, in part by the Industrial Development Fund Project of Guangzhou under Project x2jsD8183470, in part by the Engineering and Technology Research Center of Guangdong Province for Logistics Supply Chain and Internet of Things under Grant GDDST[2016]176, and in part by the Hi-Tech Industrialization Entrepreneurial Team Project of Foshan Hi-Tech Zone under Grant FSHT[2020]88.

Author Contributions All authors contributed to the study conception and methodology. [XH] and [FL] participate and guide the whole work. [JS] conducts the whole process of the experiment. The project comes from [FL]. [BW], [GZ], and [JJ] review and correctness the draft. The first draft of the manuscript was written by [Junmin Shen] and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding Information This work was supported in part by the Guangdong Major Project of Basic and Applied Basic Research under Grant 2019B030302002, in part by the Science and Technology Major Project of Guangzhou under number 202007030006, in part by the Industrial Development Fund Project of Guangzhou under Project x2jsD8183470, in part by the Engineering and Technology Research Center of Guangdong Province for Logistics Supply Chain and Internet of Things under Grant GDDST[2016]176, and in part by the Hi-Tech Industrialization Entrepreneurial Team Project of Foshan Hi-Tech Zone under Grant FSHT[2020]88.

Data availibility Not applicable.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with animals performed by any of the authors. Informed consent was obtained from all individual participants included in the study.

Informed consent For all the above contents and statements, all authors in this manuscript have informed consent.

References

1. Cusumano, M.: Cloud computing and SaaS as new computing platforms. Commun. ACM **53**(4), 27–29 (2010)
2. Gavvala, S.K., Jatoth, C., Gangadharan, G.R., Buyya, R.: QoS-aware cloud service composition using eagle strategy. Futur. Gener. Comput. Syst. **90**, 273–290 (2019)
3. Li, J., Zheng, G., Zhang, H., Shi, G.: Task scheduling algorithm for heterogeneous real-time systems based on deadline constraints. In: 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC), pp. 113–116 (2019)
4. Nathani, A., Chaudhary, S., Somani, G.: Policy based resource allocation in IaaS cloud. Futur. Gener. Comput. Syst. **28**(1), 94–103 (2012)
5. Lelong, J., Reis, V., Trystram, D.: Tuning easy-backfilling queues. In: Job Scheduling Strategies for Parallel Processing, pp. 43–61. Springer, Cham (2018)
6. Yuan, H., Liu, H., Bi, J., Zhou, M.: Revenue and energy cost-optimized biobjective task scheduling for green cloud data centers. IEEE Trans. Autom. Sci. Eng. **18**(2), 817–830 (2021)
7. Zakarya, M., Gillam, L.: Energy efficient computing, clusters, grids and clouds: a taxonomy and survey. Sustain. Comput. Inform. Syst. **14**, 13–33 (2017)
8. Elashri, S., Azim, A.: Energy-efficient offloading of real-time tasks using cloud computing. Clust. Comput. **23**(4), 3273–3288 (2020)
9. Zhu, X., Yang, L.T., Chen, H., Wang, J., Yin, S., Liu, X.: Real-time tasks oriented energy-aware scheduling in virtualized clouds. IEEE Trans. Cloud Comput. **2**(2), 168–180 (2014)
10. Bermejo, B., Juiz, C.: Virtual machine consolidation: a systematic review of its overhead influencing factors. J. Supercomput. **76**(1), 324–361 (2020)
11. Sharma, Y., Si, W., Sun, D., Javadi, B.: Failure-aware energy-efficient VM consolidation in cloud computing systems. Futur. Gener. Comput. Syst. **94**, 620–633 (2019)
12. Farahnakian, F., Pahikkala, T., Liljeberg, P., Plosila, J., Hieu, N.T., Tenhunen, H.: Energy-aware VM consolidation in cloud data centers using utilization prediction model. IEEE Trans. Cloud Comput. **7**(2), 524–536 (2019)
13. Tsai, C.-W., Huang, W.-C., Chiang, M.-H., Chiang, M.-C., Yang, C.-S.: A hyper-heuristic scheduling algorithm for cloud. IEEE Trans. Cloud Comput. **2**(2), 236–250 (2014)
14. Wang, B., Wang, C., Song, Y., Cao, J., Cui, X., Zhang, L.: A survey and taxonomy on workload scheduling and resource provisioning in hybrid clouds. Clust. Comput. **23**(4), 2809–2834 (2020)
15. Aceto, G., Botta, A., de Donato, W., Pescapè, A.: Cloud monitoring: a survey. Comput. Netw. **57**(9), 2093–2115 (2013)
16. Mahafzah, B.A., Jabri, R., Murad, O.: Multithreaded scheduling for program segments based on chemical reaction optimizer. Soft Comput. **25**(4), 2741–2766 (2021)
17. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Futur. Gener. Comput. Syst. **25**(6), 599–616 (2009)
18. Chen, X., Cheng, L., Liu, C., Liu, Q., Liu, J., Mao, Y., Murphy, J.: A WOA-based optimization approach for task scheduling in cloud computing systems. IEEE Syst. J. **14**(3), 3117–3128 (2020)
19. Sreenu, K., Sreelatha, M.: W-Scheduler: whale optimization for task scheduling in cloud computing. Clust. Comput. **22**(s1), 1087–1098 (2019)
20. Wei, X.: Task scheduling optimization strategy using improved ant colony optimization algorithm in cloud computing. J. Ambient Intell. Humaniz. Comput. (0123456789) (2020)
21. Abualigah, L., Diabat, A.: A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments. Clust. Comput. **5**, 205–223 (2020)
22. Zhou, Z., Li, F., Zhu, H., Xie, H., Abawajy, J.H., Chowdhury, M.U.: An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. Neural Comput. Appl. **32**(6), 1531–1541 (2020)
23. Iranmanesh, A., Naji, H.R.: DCHG-TS: a deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing. Clust. Comput. **24**(2), 667–681 (2021)
24. Huang, X., Li, C., Chen, H., An, D.: Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies. Clust. Comput. **23**(2), 1137–1147 (2020)
25. Kumar, M., Sharma, S.C.: PSO-COGENT: cost and energy efficient scheduling in cloud environment with deadline constraint. Sustain. Comput. Inform. Syst. **19**(January), 147–164 (2018)
26. Mishra, S.K., Puthal, D., Rodrigues, J.J.P.C., Sahoo, B., Dutkiewicz, E.: Sustainable service allocation using a metaheuristic technique in a fog server for industrial applications. IEEE Trans. Ind. Inform. **14**(10), 4497–4506 (2018)
27. Arunarani, A.R., Manjula, D., Sugumaran, V.: Task scheduling techniques in cloud computing: a literature survey. Futur. Gener. Comput. Syst. **91**, 407–415 (2019)
28. Varshney, S., Sandhu, R., Gupta, P.: Qos based resource provisioning in cloud computing environment: a technical survey. In: International conference on advances in computing and data sciences, pp. 711–723 (2019)
29. Kaur, P., Mehta, S.: Resource provisioning and work flow scheduling in clouds using augmented Shuffled Frog Leaping Algorithm. J. Parall. Distrib. Comput. **101**, 41–50 (2017)
30. Yuan, H., Zhou, M., Liu, Q., Abusorrah, A.: Fine-grained resource provisioning and task scheduling for heterogeneous

applications in distributed green clouds. IEEE/CAA J. Autom. Sin. **7**(5), 1380–1393 (2020)
31. Ding, D., Fan, X., Zhao, Y., Kang, K., Yin, Q., Zeng, J.: Q-learning based dynamic task scheduling for energy-efficient cloud computing. Futur. Gener. Comput. Syst. **108**, 361–371 (2020)
32. Aslanpour, M.S., Singh, S., Toosi, A.N.: Internet of Things Performance evaluation metrics for cloud, fog and edge computing: a review, taxonomy, benchmarks and standards for future research. Internet Things **12**, 100273 (2020)
33. Sun, H., Yu, H., Fan, G., Chen, L.: Energy and time efficient task offloading and resource allocation on the generic iot-fog-cloud architecture. Peer Peer Netw. Appl. **13**(2), 548–563 (2020)
34. Yu, H., Wang, Q., Guo, S.: Energy-efficient task offloading and resource scheduling for mobile edge computing. In: Proceeding of the IEEE International Conference Network Architecture Storage, pp. 1–4 (2018)
35. Abdullahi, M., Ngadi, M.A., Abdulhamid, S.M.: Symbiotic organism Search optimization based task scheduling in cloud computing environment. Futur. Gener. Comput. Syst. **56**, 640–650 (2016)
36. Zhang, P.Y., Zhou, M.C.: Dynamic cloud task scheduling based on a two-stage strategy. IEEE Trans. Autom. Sci. Eng. **15**(2), 772–783 (2018)
37. Masadeh, R., Sharieh, A., Mahafzah, B.: Humpback whale optimization algorithm based on vocal behavior for task scheduling in cloud computing. Int. J. Adv. Sci. Technol. **13**(3), 121–140 (2019)
38. Wu, Q., Ishikawa, F., Zhu, Q., Xia, Y., Wen, J.: Deadline-constrained cost optimization approaches for workflow scheduling in clouds. IEEE Trans. Parall. Distrib. Syst. **28**(12), 3401–3412 (2017)
39. Sahoo, S., Sahoo, B., Turuk, A.K.: A learning automata-based scheduling for deadline sensitive task in the cloud. IEEE Trans. Serv. Comput. **1374**, 1–1 (2019)
40. Masadeh, R., Alsharman, N., Sharieh, A., Mahafzah, B.A., Abdulrahman, A.: Task scheduling on cloud computing based on sea lion optimization algorithm. Int. J. Web. Inf. Syst. **17**(2), 99–116 (2021)
41. Prem Jacob, T., Pradeep, K.: A multi-objective optimal task scheduling in cloud environment using cuckoo particle swarm optimization. Wirel. Pers. Commun. **109**(1), 315–331 (2019)
42. Abdullahi, M., Ngadi, M.A., Dishing, S.I., Abdulhamid, S.M., eel Ahmad, B.I.: An efficient symbiotic organisms search algorithm with chaotic optimization strategy for multi-objective task scheduling problems in cloud computing environment. J. Netw. Comput. Appl. **133**(74), 60–74 (2019)
43. Alworafi, M.A., Mallappa, S.: A collaboration of deadline and budget constraints for task scheduling in cloud computing. Clust. Comput. **23**(2), 1073–1083 (2020)
44. Gao, Y., Wang, Y., Gupta, S.K., Pedram, M.: An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems. In: 2013 International Conference Hardware/Software Codesign System synthesizer CODES+ISSS 2013 (2013)
45. Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. **1**(1), 53–66 (1997)
46. Calheiros, R.N., Ranjan, R., Beloglazov, A., de Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software **41**(1), 23–50 (2011)
47. Li, X., Jiang, X., Garraghan, P., Wu, Z.: Holistic energy and failure aware workload scheduling in Cloud datacenters. Futur. Gener. Comput. Syst. **78**, 887–900 (2018)
48. Cohen, W.E., Mahafzah, B.A.: Statistical analysis of message passing programs to guide computer design. In: Proceedings of the thirty-first Hawaii international conference on system sciences, vol. 7, pp. 544–553 (1998). IEEE
49. Google: Google Cluster Data V2 (2011). http:// code.google.com/p/googleclusterdata/wiki/ClusterData2011_1
50. Park, K., Pai, V.S.: Comon: a mostly-scalable monitoring system for planetlab. SIGOPS Oper. Syst. Rev. **40**(1), 65–74 (2006)
51. Moreno, I.S., Garraghan, P., Townend, P., Xu, J.: Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. IEEE Trans. Cloud Comput. **2**(2), 208–221 (2014)
52. Mahafzah, B.A.: Performance evaluation of parallel multi-threaded a* heuristic search algorithm. J. Inform. Sci. **40**(3), 363–375 (2014)
53. Al-Shaikh, A., Mahafzah, B.A., Alshraideh, M.: Hybrid harmony search algorithm for social network contact tracing of COVID-19. Soft Comput. **2**, 1–23 (2021)

Xiaojian He received the Ph.D. degree in computer software and theory from Shanghai Jiao Tong University, China. He is currently an Associate Professor with the School of Computer Science and Engineering, South China University of Technology, China. His research interests include distributed computing, business intelligence, machine learning, and software development.

Fagui Liu received the M.S. degree from Beihang University and Ph.D degree from South China University of Technology in 1991 and 2006, respectively. She is currently a professor at the School of Computer Science and Engineering, South China University of Technology, China. Her current research interests include service computing, Internet of things, cloud computing and big data.

Bin Wang received his B.S. degree and Ph.D. degree from South China University of Technology in 2014 and 2021. He is currently a research scientist with the Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen. His research interests include cloud computing, edge computing and energy efficiency.

Guoxiang Zhong received the M.S. degree in the School of Applied Mathematics from Guangdong University of Technology, China in 2020. He is currently working toward the Ph.D degree at the School of Computer Science and Technology, South China University of Technology, China. His research interests include cloud computing, AIOps and machine learning.