Multi-objective Task Scheduling Optimization Based on Improved Bat Algorithm in Cloud Computing Environment

Dakun Yu, Zhongwei Xu, Meng Mei
Department of Information and Communication Engineering,
Tongji University, 201804, China

***Abstract*—In cloud computing environments, task completion time and virtual machine load balance are two critical issues that need to be addressed. To solve these problems, this paper proposes a Multi-objective Optimization Mutate Discrete Bat Algorithm (MOMDBA) that improves upon the traditional Bat algorithm (BA). The MOMDBA algorithm introduces a mutation factor and mutation inertia weight during the global optimization process to enhance the algorithm's global search ability and convergence speed. Additionally, the local optimization logic is optimized according to the characteristics of cloud computing task scenarios to improve the degree of load balancing of virtual machines. Simulation experiments were conducted using CloudSim to evaluate the algorithm's performance, and the results were compared with other scheduling algorithms. The results of our experiments indicate that when the cost difference between algorithms is within 4.47%, MOMDBA can significantly outperform other methods. Specifically, compared to PSO, GA, and LBACO, our algorithm reduces makespan by 56.26%, 59.87%, and 25.26%, respectively, while also increasing the degree of load balancing by 93.87%, 75.92%, and 39.13%, respectively. These findings demonstrate the superior performance of MOMDBA in optimizing task scheduling and load balancing.**

Keywords*—*Cloud computing; task scheduling; optimization; bat algorithm; meta-heuristics

I. Introduction

Cloud computing has become a ubiquitous infrastructure in various fields, providing users with convenient access to computing resources and services through the internet [1]. Task scheduling is a crucial problem in cloud computing that aims to allocate multiple tasks to available computing resources to optimize system throughput, response time, resource utilization, and other performance metrics. However, as cloud computing systems become increasingly complex and diverse, task scheduling problems often involve multiple conflicting optimization objectives, such as reducing system energy consumption and costs, improving task completion rates and reliability. Traditional single-objective optimization algorithms are often ineffective in solving these multi-objective optimization problems because they focus on a single objective and ignore the impact of other objectives.

To address this challenge, it is crucial to explore multi-objective optimization algorithms for cloud computing task scheduling. Multi-objective optimization algorithms can simultaneously optimize multiple objective functions and find a set of optimal solutions by balancing and trading off different objectives. These algorithms can provide more comprehensive and accurate decision support, helping cloud computing systems achieve more efficient, reliable, and sustainable operation. Additionally, they can facilitate the comprehensive evaluation and analysis of system performance, providing a more comprehensive reference for optimizing the overall performance of cloud computing systems.

The task scheduling problem is a combinatorial optimization problem that is typically considered as NP-hard [2]. Therefore, it is necessary to find an effective optimization algorithm to solve it. Traditional static task scheduling algorithms, such as Min-Min algorithm [3], Max-Min algorithm [4], and Round-Robin algorithm [5], have limitations in handling large-scale scheduling problems. Meta-heuristic algorithms [6], on the other hand, have demonstrated good robustness and feasibility in task scheduling optimization. Researchers have explored various meta-heuristic algorithms, including genetic algorithm (GA) [7], ant colony algorithm (ACO) [8], bat algorithm (BA) [9], and particle swarm optimization (PSO) algorithm [10], among others, achieving promising results [11–15].

Compared to other metaheuristic algorithms, the bat algorithm has been shown to possess strong global search capability, fast convergence speed, high search efficiency, and simple parameter settings. However, there has been limited research on the use of BA in cloud computing task scheduling, particularly with regard to multi-objective optimization. This research can effectively fill this research gap and contribute to the advancement of knowledge in this field.

Bat algorithm is a heuristic search algorithm proposed by Professor Yang in 2010[16], which is based on swarm intelligence. It is an effective method for searching the global optimal solution. The algorithm simulates the behavior of bats in nature, using a type of sonar to detect prey and avoid obstacles. This means that the bats use ultrasound to simulate the most basic detection and positioning capabilities of obstacles or prey and associate it with the optimization target function.

The bionic principle of the bat algorithm maps individual bats with the population number to NP feasible solutions in the d-dimensional problem space. The optimization process and search are simulated as the movement process of the individual bats and the search for prey. The fitness function value of the problem being solved is used to measure the quality of the position of the bat. The individual survival of the fittest process is compared to the iterative process of replacing the poor feasible solution with the good feasible solution in the optimization and search process.

The optimization principle of the bat algorithm shows that the algorithm's optimization ability primarily depends on the interaction and influence between bat individuals. However, the individuals themselves lack a mutation mechanism, which makes it difficult for them to escape from a local extreme value once they are constrained by it. Moreover, during the evolution process, super bats in a population may quickly attract other individuals to gather around them, resulting in a substantial decline in population diversity. As bat individuals get closer and closer to the optimal individuals in the population, the convergence rate is greatly reduced, or even evolutionary stagnation occurs. This causes the population to lose the ability to further evolve.

In the context of cloud computing task scheduling, this paper proposes an improved bat algorithm. By considering makespan, degree of load balancing, and cost[6] as optimization objectives, a multi-objective optimization mutation discrete bat algorithm (MOMDBA) is proposed. Building on the standard bat algorithm, the population position and velocity are discretized, and the mutation factor and mutation inertia weight are introduced to effectively balance the global search and local search ability of the algorithm, resulting in faster convergence rates. Additionally, the local optimization logic is optimized to obtain better load balancing performance based on the characteristics of the task scheduling problem.

The rest of the paper is organized as follows. Section II provides a literature review of previous works. In Section III, we model the task scheduling problem and optimization targets in the cloud computing environment. Sections IV and V introduce the bat algorithm and the proposed improved algorithm. Section VI presents the experimental results. Finally, Section VII concludes the paper.

The symbols utilized in this paper are presented in the table (Table I) below, along with their corresponding definitions.

TABLE I. Definition of Symbols Used

Symbol	Definition
$Exetime_{ij}$	The execution time of the i^{th} task (T_i) executed on the j^{th} VM (VM_j)
$length_i$	The length of the i^{th} task
$size_i$	The size of the i^{th} task
$mips_j$	Processing capacity of the j^{th} VM
bw_j	Bandwidth of the j^{th} VM
E_{VM_j}	The total execution time of VM_j
D	The degree of load balancing
f	Fitness function
x_i^t	The position of the i^{th} bat at time t
v_i^t	The speed of the i^{th} bat at time t
p_i	Mutation factor
ω	Mutation inertia weight

II. Related Work

Chen et al.[17] proposed an advanced approach called Improved WOA for Cloud task scheduling (IWC). They first mapped the task scheduling scheme to the whale foraging model to obtain an approximately optimal solution. Then, IWC was used to further improve the optimal solution search capability. The experiments demonstrate that, compared to other meta-heuristic algorithms, the proposed method has a better convergence speed and accuracy in searching for the optimal task scheduling plans.

Natesan et al.[18] proposed a modified mean grey wolf optimization algorithm that uses a variant algorithm to increase the accuracy and performance of the GWO[19]. In the proposed method, the encircling equation and hunting equation were modified to improve the efficiency of the motion, and a suitable path for each wolf was present in the searching area. The modifications to the GWO algorithm led to improved convergence speed and accuracy in solving the task scheduling problem in cloud computing.

Jacob et al.[20] combined two optimization algorithms, Cuckoo Search and Particle Swarm Optimization, to reduce the makespan, cost, and deadline violation rate. The newly proposed hybrid algorithm is called CPSO. From the simulation results, the proposed method outperforms PBACO, ACO, MIN-MIN, and FCFS in terms of minimizing the makespan, cost, and deadline violation rate.

Jing et al.[21] proposed a QoS-aware cloud task scheduling algorithm called QoS-DPSO, which aims to satisfy the QoS requirements in cloud computing systems. They took into account the user's preference for QoS requirements and considered enough QoS parameters. The proposed method obtained superior performance by incorporating QoS requirements into the task scheduling process.

Kumar et al.[22] proposed a hybrid multi-objective optimization algorithm called HGA-ACO. They combined Ant Colony Optimization (ACO) algorithm with the Genetic algorithm (GA) to obtain better performance in task allocation. ACO is used to assist GA in avoiding local optimal solutions, while GA is used to enhance the ACO solutions. The proposed hybrid algorithm exhibits better performance in terms of task allocation compared to other existing algorithms.

Hamad et al.[23] proposed a Genetic-Based task scheduling algorithm to minimize the completion time and cost of tasks, and to maximize resource utilization. According to the experiments, the completion time and cost for the proposed method were reduced by 41.83% and 3.6%, respectively, compared to the standard GA. Additionally, the resource utilization was improved by 47%. The proposed algorithm shows potential for improving the efficiency of task scheduling in cloud computing systems.

Wei et al.[24] proposed an improved ant colony algorithm to solve the problem of unbalanced task scheduling load and low reliability in cloud computing environments. They improved the pheromone update and pheromone volatilization methods for the ant colony algorithm to speed up the convergence speed and introduced the load weight coefficient of VM in the update process of local pheromone. Experimental results verify the feasibility of the proposed method, which reduces the task scheduling completion time and convergence time while ensuring load balancing. The proposed method shows potential for improving the performance of task scheduling in cloud computing systems.

From previous studies, it can be concluded that the improvement of meta-heuristic algorithms can be generally divided into two types: one is to improve on the basis of classical algorithms, and the other is to combine two algorithms to form a new algorithm. The proposed method belongs to the first type, which is based on the bat algorithm and finds the optimal task scheduling scheme by mutation while randomly initializing the bat population position. Additionally, the local optimization logic of the bat algorithm is optimized to improve the algorithm's load balancing performance.

III. Resource Scheduling Model in Cloud Environment

The resource scheduling model in the cloud environment is shown in Fig. 1. In this model, resource scheduling is divided into two levels. The first level is task to virtual machine scheduling, and the second level is virtual machine to host scheduling, which assigns virtual machines to different physical hosts in the data center. This paper mainly focuses on the task scheduling process at the first level. In the task scheduling process, the scheduling algorithm first segments the tasks submitted by users, which is a complicated process in actual operation. For the convenience of research, this process is idealized in this paper, where tasks are assumed to be independent of each other. Tasks are executed in no particular sequence, and cannot be interrupted or migrated. The computing capacity of all computing resources is known. In this paper, makespan, degree of load balancing, and cost are taken as optimization objectives of the algorithm.

A. Makespan

Makespan defines the total time required from submitting a task to the completion of the task by the user[25].

With m tasks, $Task = \{T_1, T_2, \ldots, T_m\}$ and n VMs, $VM = \{VM_1, VM_2, \ldots, VM_n\}$ where $m > n$, assuming that a subtask can run on only one VM, the mapping between a subtask and a VM is by TV_{map}.

$$TV_{map} = \begin{Bmatrix} T_1V_1 & T_1V_2 & \ldots & T_1V_n \\ T_2V_1 & T_2V_2 & \ldots & T_2V_n \\ \vdots & \vdots & & \vdots \\ T_mV_1 & T_mV_2 & \ldots & T_mV_n \end{Bmatrix} \tag{1}$$

The execution time of the i^{th} task (T_i) executed on the j^{th} VM (VM_j) is denoted by $Exetime_{ij}$, and can be calculated as follows:

$$Exetime_{ij} = \frac{length_i}{mips_j} + \frac{size_i}{bw_j} \tag{2}$$

where $length_i$ and $size_i$ represents the length and the size of the i^{th} task, and $mips_j$ and bw_j refer to the processing capacity and bandwidth of the j^{th} VM. If E_{VM_j} is the total execution time of VM_j, then:

$$E_{VM_j} = \sum_{i \in I} Exetime_{ij} \tag{3}$$

where I is the set of subtasks executed on VM_j. Then, the makespan (E) is defined as the maximum total execution time among all VMs:

$$E = max\{E_{VM_1}, E_{VM_2}, \ldots, E_{VM_n}\} \tag{4}$$

Therefore, one goal of the task scheduling problem is to minimize the makespan, which represents the total time required for all tasks to complete their execution.

B. Degree of Load Balancing

The degree of load balancing is an essential indicator that describes the current working status of VMs in a cloud computing system. It measures the degree to which the available resources of VMs are utilized. A higher degree of load balancing indicates that the resources of VMs are fully utilized, and the workload is distributed equally among all VMs. In contrast, a lower degree of load balancing indicates that some VMs are overloaded, while others are underutilized. Therefore, achieving a high degree of load balancing is crucial for enhancing the efficiency and performance of cloud computing systems.

In a task scheduling scheme, the execution time of VM_j is denoted by E_{VM_j}, and the maximum execution time of all VMs is denoted by $E_{VM_{max}}$.

The degree of load balancing in this task allocation scheme can be expressed as follows:

$$D = \frac{1}{n} \sum_{j=1}^{n} \frac{E_{VM_j}}{E_{VM_{max}}} \tag{5}$$

This equation represents the average ratio of the execution time of each VM to the maximum execution time among all VMs. A higher value of the degree of load balancing indicates that the workload is evenly distributed among all VMs and that the available resources are fully utilized. Therefore, one goal of task scheduling is to maximize the degree of load balancing, which ultimately improves the efficiency and performance of the cloud computing system.

C. Cost

Cost is the sum of the costs used by VMs to execute tasks in a task scheduling scheme. It can be calculated as follows:

$$Cost = \sum_{j=1}^{n} E_{VM_j} C_j \tag{6}$$

where E_{VM_j} is the total execution time of VM_j and C_j is the cost per second of VM_j. This equation represents the total cost incurred by all VMs in the task scheduling scheme. The cost per second of each VM is determined by various factors, such as the processing power, memory capacity, and network bandwidth. Therefore, minimizing the cost is also an important objective of task scheduling, as it leads to the efficient utilization of resources and reduces the overall cost of the cloud computing system.

D. Multiobjective Fitness Function

In this paper, the linear weighting method is used to transform the multi-objective optimization problem into a single

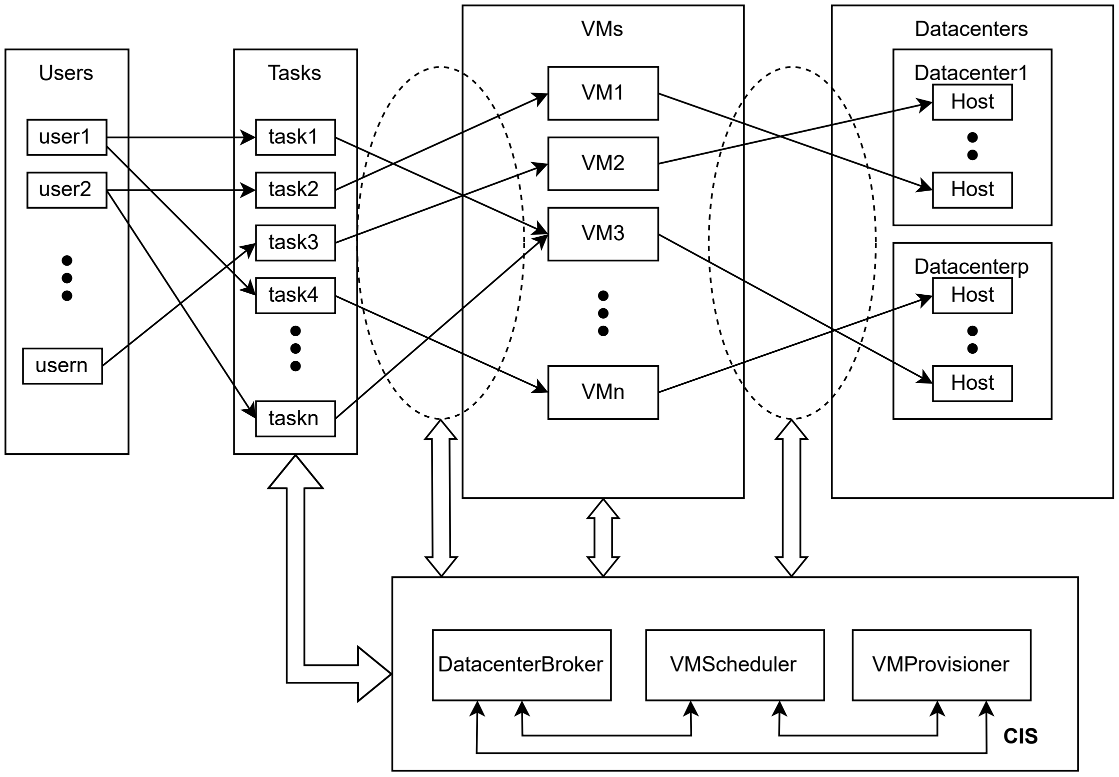

Fig. 1. Task schedule model in cloud environment.

objective optimization problem. Since the different optimization objectives have different dimensions, it is necessary to normalize them. The normalization process is shown below:

$$T_s = \frac{E_s}{m} \tag{7}$$

$$f_T = \frac{T_s - T_{s,min}}{T_{s,max} - T_{s,min}} \tag{8}$$

$$C_s = \frac{Cost_s}{nE_sD_s} \tag{9}$$

$$f_C = \frac{C_s - C_{s,min}}{C_{s,max} - C_{s,min}} \tag{10}$$

Here, f_T represents the normalized task execution time indicator, where E_s is the maximum execution time using the current task scheduling scheme, T_s represents the average execution time of one task. f_C represents the normalized task cost indicator, where D_s is the degree of load balancing using the current task scheduling scheme, $Costs_s$ is the cost using the current task scheduling scheme, and C_s is the average cost per second per virtual machine.

The fitness function used in this paper is defined as follows:

$$f = \alpha f_T + \beta f_C \tag{11}$$

where $\alpha \in [0, 1]$, $\beta \in [0, 1]$, and $\alpha + \beta = 1$. The optimization objective of this paper is to minimize the fitness function, which is a linear combination of the normalized task execution time indicator and the normalized task cost indicator. The weights α and β can be adjusted to give different importance to the two objectives based on the requirements of the cloud computing system.

Overall, the goal of the task scheduling problem is to find the optimal task allocation scheme that minimizes the makespan, maximizes the degree of load balancing, and minimizes the cost of the system. By transforming the multi-objective optimization problem into a single objective optimization problem and using the fitness function, this paper provides a framework for achieving these goals through task scheduling.

IV. Standard Bat Algorithm

The bat algorithm is a swarm intelligence optimization algorithm that mimics the echolocation behavior of bats in nature to find the optimal solution in a given search space. The optimization process is completed through a series of iterations, where the positions of the bats in the population are updated to improve the quality of the candidate solutions.

In the bat algorithm, each bat is represented by a position vector in a d-dimensional search space, and is randomly initialized with an initial position and velocity. At each iteration, the bats adjust their positions based on their current location and the global best solution found so far. The update formula for the position and velocity of each bat is:

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{12}$$

$$v_i^{t+1} = v_i^t + (x_i^t - x^*)f_i \tag{13}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{14}$$

where β is a random number between 0 and 1, and x^* is the current optimal population solution.

The bat algorithm also includes a local search mechanism to promote exploration of the search space. In the local search, each bat generates a new position by randomly walking around the current global best solution. The new position is calculated using the formula:

$$x_{new} = x_{old} + \delta A^t \tag{15}$$

where x_{old} is the current global best solution, x_{new} is the new local solution, and δ is a randomly generated d-dimensional vector in the range $[-1, 1]$. A^t represents the current average pulse loudness of the entire population.

The bat algorithm also introduces randomness in the search process by allowing each bat to choose between a global search and a local search with a certain probability. The probability of choosing a global search is controlled by the parameter r_i, which is initialized to a small value and gradually increases during the search process. If a bat generates a random number R greater than r_i, it performs a local search near the global best solution, and if R is less than r_i, it performs a global search.

Finally, the bat algorithm updates the pulse loudness and pulse rate of each bat based on its performance in the search process. If a bat finds a better solution, its pulse loudness decreases, and its pulse rate increases, which increases the bat's ability to exploit the promising regions of the search space. The updated pulse loudness and pulse rate are then used in the next iteration to adjust the frequency and loudness of the bat's pulse. The update formula for the pulse loudness and pulse rate is:

$$A_i^{t+1} = \alpha A_i^t \tag{16}$$

$$r_i^{t+1} = r_i^0(1 - exp(-\gamma t)) \tag{17}$$

where α and γ are constants with $0 < \alpha < 1$ and $\gamma > 0$. The parameter A_i represents the pulse loudness of the ith bat, and r_i represents the pulse rate. The pulse loudness is decreased by multiplying it with the constant α when a better solution is found, and the pulse rate is updated based on the current iteration number t using the formula given in Eq. (17).

In summary, the bat algorithm is a powerful optimization algorithm that combines global and local search strategies to efficiently explore the search space and find the optimal solution. The algorithm has been successfully applied to a wide range of optimization problems in various fields, including engineering, finance, and computer science. Its effectiveness and robustness make it a popular choice among researchers and practitioners in optimization and swarm intelligence.

V. MOMDBA-Multi-objective Mutated Discrete Bat Algorithm

Similar to other meta-heuristic algorithms, bat algorithm also has the problem that it is easy to fall into local optimization and the global search ability is insufficient.

According to the characteristics of task scheduling problem, this paper introduces mutation factor, mutation inertia weight and optimization of local optimization logic to optimize the standard bat algorithm and improve the algorithm performance.

A. Discretization Coding

The MOMDBA algorithm uses the position of each bat to represent a feasible task scheduling scheme, which is a viable solution to the optimization objective. It assumes that the number of available virtual machines in the cloud platform is fixed, and the number of cloud tasks that need to be processed is constant. Therefore, the position of the i^{th} bat at time t can be expressed as an m-dimensional vector $x_i^t = (VMtask_1, VMtask_2, \ldots, VMtask_m)$, where $VMtask_i$ indicates the virtual machine that executes the i^{th} task.

The speed of each bat $v_i = (v_{i1}, v_{i2}, \ldots, v_{im})$ represents the change in the virtual machines assigned to each cloud task from the current schedule scheme to the new schedule scheme. The dimension of v_i^t is the same as the dimension of the position, and its update formula is given by:

$$v_i^t = f(x_i^t - x^*) \tag{18}$$

where x^* is the best solution found so far, and $f(v_{ik})$ is a function that determines the direction of change for the k^{th} dimension of the speed. Specifically, $f(v_{ik})$ is defined as follows:

$$f(v_{ik}) = \begin{cases} 1 & \text{if } v_{ik} > 0 \\ 0 & \text{if } v_{ik} = 0 \end{cases} \tag{19}$$

Here, v_{ik} refers to the speed of the bat in the k^{th} dimension, where $k \in (0, m]$.

B. Mutation Factor and Mutation Inertia Weight

The standard bat algorithm updates the bat's optimization direction based on the current bat position and the current optimal solution. However, this approach is not suitable for the task scheduling problem because there is no correlation between different virtual machines. To address this issue, this paper introduces a mutation method to optimize the task scheduling scheme, which includes a mutation factor $p_i = (p_{i1}, p_{i2}, \ldots, p_{im})$ and a mutation inertia weight ω.

In the proposed method, the bat's position is updated randomly, and the probability of position update is determined by the mutation factor and the speed of the bat. Specifically, the probability of selecting a random virtual machine for the task is given by:

$$P(x_{ik}^{t+1} = Random) = p_{ik} \tag{20}$$

The probability of selecting the current optimal solution for the task is given by:

$$P(x_{ik}^{t+1} = x_k^*) = \theta v_{ik}^t p_{ik} \tag{21}$$

And the probability of keeping the current position is given by:

$$P(x_{ik}^{t+1} = x_{ik}^t) = 1 - p_{ik} - \theta v_{ik}^t p_{ik} \tag{22}$$

Here, x_{ik}^t refers to the position of the bat in the k^{th} dimension at time t, $k \in (0, m]$, and p_{ik} is the mutation factor of the k^{th} dimension, with values in the range of $(0, 0.5)$. θ is a constant and $0 < \theta < 1$.

To balance the local and global optimization ability of the bats, the mutation probability is adjusted after a bat finds a better solution, using the mutation inertia weight ω. The mutation factor p_{ik} is updated as follows:

$$\begin{cases} p_{ik}^{t+1} = p_{ik}^t \omega & \text{if } x_{ik}^{t+1} = x_{ik}^t \\ p_{ik}^{t+1} = p_{ik}^t & \text{if } x_{ik}^{t+1} \neq x_{ik}^t \end{cases} \tag{23}$$

The mutation inertia weight ω is a function of time, and its value decreases as the number of iterations increases, helping the algorithm to converge quickly. Specifically, ω is given by:

$$\omega = \omega^0(1 - exp(-\lambda t)) \tag{24}$$

Here, λ is a constant parameter with values in the range of $(0, 1)$, ω^0 is the initial value of the mutation inertia weight, and ω is constrained to the range of $(0, 1)$. When the number of iterations is small, the mutation probability of bats is large. In this case, the global search ability of the algorithm is strong, which is conducive to jumping out of the local optimal solution and obtaining the global optimal solution. When the number of iterations is small, the mutation probability of the bats is high, which enhances the global search ability of the algorithm and helps it to jump out of local optimal solutions. When a bat finds a better solution and the new solution in the k^{th} dimension is the same as the old solution, the mutation factor p_{ik} is updated based on the mutation inertia weight ω. However, if the new solution is different from the old solution, the mutation factor remains unchanged. By adjusting the mutation probability using the mutation inertia weight, the proposed method can balance the exploration and exploitation phases and improve the overall performance of the bat algorithm for task scheduling problems.

C. Local Optimization Logic

In the proposed method, the execution time of each virtual machine (VM) is denoted by $T = \{E_{VM_1}, E_{VM_2}, \ldots, E_{VM_n}\}$, where $E_{VM_{max}}$ and $E_{VM_{min}}$ are the maximum and minimum execution times, respectively. Let $Exetime_{kmax}$ and $Exetime_{kmin}$ be the execution times of the cloud task k on VM_{max} and VM_{min}, respectively. During local optimization, the following update rules are applied:

$$E_{VM_{max}}^{t+1} = E_{VM_{max}}^t - Exetime_{kmax} \tag{25}$$

$$E_{VM_{min}}^{t+1} = E_{VM_{min}}^t + Exetime_{kmin} \tag{26}$$

$$x_{newk} = VM_{min} \tag{27}$$

Here, x_{newk} is the k^{th} dimension position of the current global optimal solution. The goal of local optimization is to improve the degree of load balancing by reassigning the task k from VM_{max} to VM_{min}.

Before updating the bat population, a random number in the range of $[0, 1]$ is generated. If the random number R is greater than the pulse emission rate r_i of the i^{th} bat, then local optimization is performed; otherwise, global optimization is performed. The pulse emission rate of each bat is updated using the same formula (Eq.(17)) as in the standard bat algorithm.

D. The steps of MOMDBA

The proposed method consists of the following steps:

- **Step 1:** Initialize the bat population by randomly scheduling m tasks to n virtual machines. The dimension of the bat location x_i is the number of cloud tasks m.
- **Step 2:** Generate a random number R. If $R < r_i$, go to Step 3; otherwise, go to Step 4.
- **Step 3:** Update the bat positions according to mutation factors. The position of each bat is updated based on the current position, the current optimal solution, and the mutation probability. If a better solution is found, update r_i and p_i; otherwise, keep them unchanged. Go to Step 5.
- **Step 4:** Update the bat locations according to a local optimization logic. Calculate the execution times of the tasks on each virtual machine and reassign the tasks to achieve load balancing. Go to Step 5.
- **Step 5:** Check if a better solution is found. If yes, update r_i and p_i based on the current mutation probability and the mutation inertia weight. If no, keep r_i and p_i unchanged. Go to Step 6.
- **Step 6:** If the current iteration times are less than the maximum number of iterations, go back to Step 1 and repeat the process; otherwise, go to Step 7.
- **Step 7:** Output the optimal bat location as the best task schedule scheme.

VI. Experiment and Result

To verify the effectiveness of MOMDBA in solving cloud computing task scheduling problems, the proposed method is simulated using CloudSim and compared with other existing algorithms, including PSO, GA, and LBACO[26], using the publicly available GoCJ dataset proposed by Hussain et al.[27]. The performance of the proposed algorithm is evaluated based on four criteria: fitness function value, makespan, degree of load balancing, and cost.

The GoCJ dataset consists of 19 text files containing task lengths ranging from 100 to 1000. Each line in the text file corresponds to the task length of a task, and the tasks are classified into five types based on their length: small, medium, large, extra-large, and huge. The distribution of each task in the dataset is shown in Table II.

The basic steps of CloudSim simulation are as follows:

- **Step 1:** Initialize CloudSim.
- **Step 2:** Instantiate the DataCenter, DataCenterBroker, and virtual machines.
- **Step 3:** Create a list of virtual machines and register them with the DataCenter. Then, submit the list of

virtual machines to the DataCenterBroker for further management and scheduling.

- **Step 4:** Generate a set of tasks and assign them to the DataCenterBroker for further processing. The DataCenterBroker is responsible for managing and scheduling the tasks on available virtual machines in the data center.

- **Step 5:** Start the simulation.

- **Step 6:** Analyzing the simulation results.

TABLE II. Job types of GoCJ

Job type	MI range	Distribution
Small	15,000 - 55,000	20%
Medium	59,000-99,000	40%
Large	101,000-135,000	30%
Extra large	150,000-337,000	6%
Huge	525,000-900,000	4%

A. Experimental Environment Setting

The experiments in this paper were conducted on a personal computer environment. The detailed configurations of the software and hardware environments are shown in Table III.

The number of virtual machines used in the experiments was set to 15, divided into three groups of low, medium, and high performance, with five virtual machines in each group. The information of the virtual machines is shown in Table IV. The parameter settings of each algorithm used in the experiments are shown in Table V.

TABLE III. Simulation Environment

Parameter	Configuration
CPU	AMD Ryzen5 5600X
Memory	16GB
Hard disk	1TB
IDE	IntelliJ IDEA 2022.03

TABLE IV. Virtual Machine Information

VM group	VM ID	VM performance/MIPS	Cost per sec
low performance	0-4	800-1200	0.02
medium performance	5-9	1800-2200	0.06
high performance	10-14	3800-4200	0.13

TABLE V. Parameters Setting

Algorithm	Parameter	Value
MOMDBA	Population size	50
	Maximum iterations	200
	λ	0.01
	θ	0.8
	γ	0.08
	r^0	0.8
	ω^0	0.7
PSO	Population size	50
	Maximum iterations	200
	ω	0.9
	c_1	2.0
	c_2	2.0
GA	Population size	50
	Maximum iterations	200
	p_c	0.8
	p_m	0.01
LBACO	Population size	50
	Maximum iterations	200
	ρ	0.5
	α	2.0
	β	1.0
	Q	100

B. Result

In order to evaluate the performance of MOMDBA, algorithm simulation was conducted on the GoCj data set, with the number of cloud tasks ranging from 100 to 500, and analyzed from four aspects of fitness, makespan, degree of load balancing and cost, and compared with other meta-heuristic algorithms.

The fitness function used in the experiments is defined as Eq. (11), which represents the comprehensive evaluation index of multi-objective optimization. The lower the fitness value, the better the solution found by the algorithm. Fig. 2 shows that the fitness value of MOMDBA is significantly lower than that of other meta-heuristic algorithms under different circumstances.

Table VI shows a reduction in fitness using GoCJ dataset. The data show that MOMDBA is up to 80% less fitness than PSO, 84% less fitness than GA, and 70% less fitness than LBACO. he introduction of mutation factor and mutation inertia weight enables the proposed algorithm to jump out of local optimal solutions and obtain better optimization capability.

Fig. 3 and Fig. 4 show the makespan and cost of each algorithm under different number of cloud tasks, and the specific values are detailed in Tables VII and VIII. It can be seen that with the increase in the number of cloud tasks, the makespan and cost of the algorithms also increase gradually.

Regarding makespan, MOMDBA achieves significantly less makespan than the other algorithms, with a maximum reduction of 56.26% compared to PSO, 59.87% compared to GA, and 25.26% compared to LBACO. This indicates

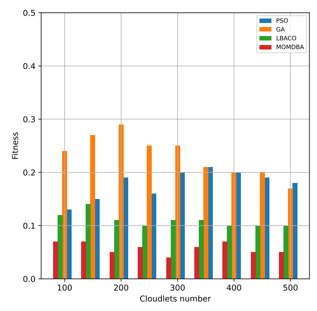

Fig. 2. Fitness based experimental results.

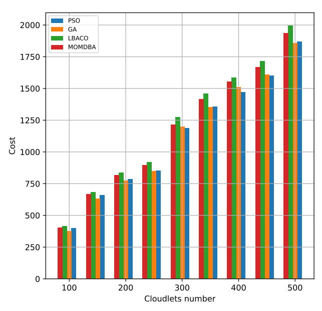

Fig. 4. Cost based experimental results.

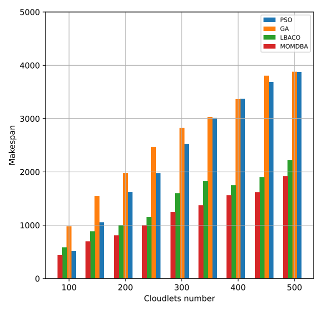

Fig. 3. Makespan based experimental results.

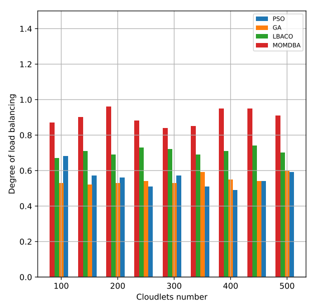

Fig. 5. Degree of load balancing based experimental results.

that MOMDBA can significantly reduce makespan, which can improve the efficiency of the cloud platform and can handle more tasks in less time.

For cost, there is no significant difference between MOMDBA and the other algorithms. Except for reducing the cost of LBACO by a maximum of 4.47%, MOMDBA is 8% higher than GA and 5.52% higher than PSO, respectively. This is acceptable compared to the significant reductions made by makespan.

Regarding the degree of load balancing. Fig. 5 and Table IX show how MOMDBA differs from other algorithms. According to the data in the table, the degree of load balancing of MOMDBA is up to 93.87% higher than PSO, 75.92% higher than GA, and 39.13% higher than LBACO. MOMDBA has excellent load balancing, which benefits from the improved local optimization logic. It allows MOMDBA to maintain a high degree of load balancing even when facing a large number of cloud tasks.

In summary, MOMDBA has stronger optimization performance, less makespan, and higher degree of load balancing than other meta-heuristic algorithms. The introduction of mutation factor and mutation inertia weight enables MOMDBA to achieve better optimization capabilities by jumping out of local optimal solutions. These results demonstrate the effectiveness of the proposed algorithm in solving cloud computing task scheduling problems.

VII. Conclusion

This paper proposes an improved bat algorithm for task scheduling. Based on the original bat algorithm, the mutation factor and mutation inertia weight are introduced, and the logic of local optimization is enhanced. The proposed method is simulated using CloudSim and compared with other meta-heuristic algorithms on a public dataset. The performance of the proposed method is analyzed from four perspectives: fitness, makespan, cost, and degree of load balancing. Experimental results demonstrate that the proposed algorithm has stronger optimization ability and can consistently achieve lower fitness scores. In the case of similar costs, the proposed algorithm outperforms other algorithms in terms of makespan and load balancing.

While MOMDBA performs well in terms of makespan and load balancing, it does not provide a significant cost advantage over other algorithms. In our future studies, we plan to further optimize the objective function and explore more effective approaches to solving cloud computing task scheduling problems in complex scenarios. This may involve considering additional factors such as memory and bandwidth constraints. Additionally, we aim to investigate the potential benefits of combining MOMDBA with deep reinforcement learning techniques to further enhance the algorithm's performance.

By addressing these challenges, we hope to improve the overall service performance of cloud systems and provide more efficient and cost-effective solutions for cloud computing applications.

TABLE VI. FITNESS BASED EXPERIMENTAL RESULTS.

Cloudlets	LBACO	GA	PSO	MOMDBA	Reduction in Fitness using MOMDBA		
					Over LBACO	Over GA	Over PSO
100	0.12	0.24	0.13	0.07	41.66%	70.83%	46.15%
150	0.14	0.27	0.15	0.07	50.00%	74.07%	53.33%
200	0.11	0.29	0.19	0.05	54.54%	82.75%	73.68%
250	0.10	0.25	0.16	0.06	60.00%	76.00%	62.50%
300	0.11	0.25	0.20	0.04	63.63%	84.00%	80.00%
350	0.11	0.21	0.21	0.06	45.45%	71.42%	71.42%
400	0.10	0.20	0.20	0.07	70.00%	65.00%	65.00%
450	0.10	0.20	0.19	0.05	50.00%	75.00%	73.68%
500	0.10	0.17	0.18	0.05	50.00%	70.58%	72.22%

TABLE VII. MAKESPAN BASED EXPERIMENTAL RESULTS

Cloudlets	LBACO	GA	PSO	MOMDBA	Reduction in Makespan using MOMDBA		
					Over LBACO	Over GA	Over PSO
100	577	975	513	441	23.57%	54.77%	14.03%
150	886	1552	1047	697	21.33%	55.09%	33.42%
200	1003	1978	1620	811	19.14%	58.99%	49.93%
250	1151	2470	1972	991	13.90%	59.87%	49.76%
300	1596	2826	2522	1251	21.61%	55.73%	50.39%
350	1829	3026	3014	1367	25.26%	54.82%	54.64%
400	1750	3366	3374	1562	10.74%	53.59%	53.70%
450	1901	3801	3686	1612	15.20%	57.59%	56.26%
500	2216	3878	3868	1919	13.40%	50.51%	50.38%

TABLE VIII. COST BASED EXPERIMENTAL RESULTS

Cloudlets	LBACO	GA	PSO	MOMDBA	Reduction in Cost using MOMDBA		
					Over LBACO	Over GA	Over PSO
100	415	375	399	405	2.40%	-8.00%	-1.50%
150	684	632	658	667	2.48%	-5.53%	-1.37%
200	837	773	786	816	2.50%	-5.56%	-3.82%
250	919	850	851	898	2.29%	-5.64%	-5.52%
300	1274	1201	1186	1217	4.47%	-1.33%	-2.61%
350	1458	1352	1356	1418	2.74%	-4.88%	-4.57%
400	1586	1510	1473	1555	1.95%	-2.98%	-5.27%
450	1715	1608	1602	1668	2.74%	-3.73%	-4.12%
500	1997	1856	1870	1936	3.01%	-4.31%	-3.53%

TABLE IX. DEGREE OF LOAD BALANCING BASED EXPERIMENTAL RESULTS

Cloudlets	LBACO	GA	PSO	MOMDBA	Improvment in degree of load balacing using MOMDBA		
					Over LBACO	Over GA	Over PSO
100	0.67	0.53	0.68	0.87	29.85%	64.15%	27.94%
150	0.71	0.52	0.57	0.90	26.76%	73.07%	57.89%
200	0.69	0.53	0.56	0.96	39.13%	44.79%	71.42%
250	0.73	0.54	0.51	0.88	20.54%	62.96%	72.54%
300	0.72	0.53	0.57	0.84	14.28%	58.49%	47.36%
350	0.69	0.59	0.51	0.85	23.18%	44.06%	66.67%
400	0.71	0.55	0.49	0.95	33.80%	72.72%	93.87%
450	0.74	0.54	0.54	0.95	38.34%	75.92%	75.92%
500	0.70	0.60	0.59	0.91	30.00%	51.67%	54.23%

ACKNOWLEDGMENT

The authors would like to thank the support of the National Key Research and Development Program of China (2022YFB4300504-4).

REFERENCES

[1] T. Taami, S. Krug, and M. O'Nils, "Experimental characterization of latency in distributed iot systems with cloud fog offloading," in *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2019, pp. 1–4.

[2] V. Hayyolalam, B. Pourghebleh, M. R. Chehrehzad, and A. A. Pourhaji Kazem, "Single-objective service composition methods in cloud manufacturing systems: Recent techniques, classification, and future trends," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 5, p. e6698, 2022.

[3] S. S. Murad, R. Badeel, N. Salih, A. Alsandi, R. Faraj, A. Ahmed, A. Muhammed, M. Derahman, and N. Alsandi, "Optimized min-min task scheduling algorithm for

scientific workflows in a cloud environment," *J. Theor. Appl. Inf. Technol*, vol. 100, pp. 480–506, 2022.

[4] O. Elzeki, M. Reshad, and M. A. Elsoud, "Improved max-min algorithm in cloud computing," *International Journal of Computer Applications*, vol. 50, no. 12, 2012.

[5] F. Alhaidari and T. Z. Balharith, "Enhanced round-robin algorithm in the cloud computing environment for optimal task scheduling," *Computers*, vol. 10, no. 5, p. 63, 2021.

[6] Z. Beheshti and S. M. H. Shamsuddin, "A review of population-based meta-heuristic algorithms," *Int. J. Adv. Soft Comput. Appl*, vol. 5, no. 1, pp. 1–35, 2013.

[7] K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam, "A genetic algorithm (ga) based load balancing strategy for cloud computing," *Procedia Technology*, vol. 10, pp. 340–347, 2013.

[8] Y. Natarajan, S. Kannan, and G. Dhiman, "Task scheduling in cloud using aco," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, vol. 15, no. 3, pp. 348–353, 2022.

[9] L. Jacob, "Bat algorithm for resource scheduling in cloud computing," *population*, vol. 5, no. 18, p. 23, 2014.

[10] S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, "Heuristic initialization of pso task scheduling algorithm in cloud computing," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 2370–2382, 2022.

[11] P. Singh, M. Dutta, and N. Aggarwal, "A review of task scheduling based on meta-heuristics approach in cloud computing," *Knowledge and Information Systems*, vol. 52, pp. 1–51, 2017.

[12] R. M. Singh, S. Paul, and A. Kumar, "Task scheduling in cloud computing," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 6, pp. 7940–7944, 2014.

[13] A. Razaque, N. R. Vennapusa, N. Soni, G. S. Janapati *et al.*, "Task scheduling in cloud computing," in *2016 IEEE long island systems, applications and technology conference (LISAT)*. IEEE, 2016, pp. 1–5.

[14] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: review, taxonomy, open challenges, and future trends," *Swarm and Evolutionary Computation*, vol. 62, p. 100841, 2021.

[15] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Generation Computer Systems*, vol. 91, pp. 407–415, 2019.

[16] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pp. 65–74, 2010.

[17] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, and J. Murphy, "A woa-based optimization approach for task scheduling in cloud computing systems," *IEEE Systems Journal*, vol. 14, no. 3, pp. 3117–3128, 2020.

[18] G. Natesan and A. Chokkalingam, "Task scheduling in heterogeneous cloud environment using mean grey wolf optimization algorithm," *ICT Express*, vol. 5, no. 2, pp. 110–114, 2019.

[19] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in engineering software*, vol. 69, pp. 46–61, 2014.

[20] T. Prem Jacob and K. Pradeep, "A multi-objective optimal task scheduling in cloud environment using cuckoo particle swarm optimization," *Wireless Personal Communications*, vol. 109, pp. 315–331, 2019.

[21] W. Jing, C. Zhao, Q. Miao, H. Song, and G. Chen, "Qos-dpso: Qos-aware task scheduling for cloud computing system," *Journal of Network and Systems Management*, vol. 29, pp. 1–29, 2021.

[22] A. Senthil Kumar and M. Venkatesan, "Multi-objective task scheduling using hybrid genetic-ant colony optimization algorithm in cloud environment," *Wireless Personal Communications*, vol. 107, pp. 1835–1848, 2019.

[23] S. A. Hamad and F. A. Omara, "Genetic-based task scheduling algorithm in cloud computing environment," *Int. J. Adv. Comput. Sci. Appl*, vol. 7, no. 4, pp. 550–556, 2016.

[24] X. Wei, "Task scheduling optimization strategy using improved ant colony optimization algorithm in cloud computing," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–12, 2020.

[25] D. Alsadie, "Tsmgwo: Optimizing task schedule using multi-objectives grey wolf optimizer for cloud data centers," *IEEE Access*, vol. 9, pp. 37 707–37 725, 2021.

[26] Z. Huan-qing, Z. Xue-ping, W. Hai-tao, and L. Yan-han, "Task scheduling algorithm based on load balancing ant colony optimization in cloud computing," *Microelectronics and computer*, vol. 32, no. 5, pp. 31–35, 2015.

[27] A. Hussain and M. Aleem, "Gocj: Google cloud jobs dataset for distributed and cloud computing infrastructures," *Data*, vol. 3, no. 4, p. 38, 2018.