An Improved Coral Reef Optimization-Based Scheduling Algorithm for Cloud Computing

Shuzhen Wan$^1$ and Lixin Qi$^2$

$^1$Hubei Key Laboratory of Intelligent Vision Based Monitoring for Hydroelectric Engineering, China Three Gorges University, Hubei, Yichang, China
$^2$College of Computer and Information Technology, China Three Gorges University, Hubei, Yichang, China

Correspondence should be addressed to Shuzhen Wan; wanshuzhen@163.com

Received 28 February 2021; Accepted 10 July 2021; Published 26 July 2021

Abstract

An important problem in cloud computing faces the challenge of scheduling tasks to virtual machines to meet the cost and time demands, while maintaining the Quality of Service (QoS). Allocating tasks into cloud resources is a difficult problem due to the uncertainty of consumers’ future requirements and the diversity of providers’ resources. Previous studies, either on modeling or scheduling approaches, cannot offer a satisfactory solution. In this paper, we establish a resource allocation framework and propose a novel task scheduling algorithm. An improved coral reef optimization (ICRO) is proposed to deal with this task scheduling problem. In ICRO, the better-offspring and multicrossover strategies increase the convergent speed and improve the quality of solutions. In addition, a novel load balance-aware mutation enhances the load balance among virtual machines and adjusts the number of resources provided to users. Experimental results show that compared with other algorithms, ICRO can significantly reduce the makespan and cost of the scheduling, while maintaining a better load balance in the system.

1. Introduction

Cloud computing [1, 2] has brought great benefits to individuals and enterprises in recent years. It provides large-scale and flexible utility computing services to consumers with a good scalability and the reduction of hardware and software cost.

Cloud services can be classified into three levels: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [3, 4]. IaaS is a computing service type widely employed in the cloud computing paradigm. In this framework, virtualization technologies are applied to provide resources to cloud users by creating virtual machines (VMs) on limited hardware. Cloud users can access almost unlimited computing resources at a low cost [5]. The computing services providers, such as IBM Smart Cloud Enterprise [1] and Amazon EC2 [6], are responsible for guaranteeing the Quality of Service (QoS) for running the VMs. This QoS may be defined by different Service Level Agreements (SLAs) which can specify the QoS requirements of negotiation resources, minimum limits, and expectations between providers and consumers [7]. Service providers can offer different types of VMs that are characterized by machine configurations, QoS, and price model. In this model of service, consumers can easily subscribe VMs to execute their tasks; on the contrary, efficiently scheduling a huge number of tasks is a critical concern for cloud providers.

In cloud computing environments, resource management of the IaaS layer is responsible for scheduling tasks for execution. Scheduling tasks to VMs is the key issue on the IaaS layer because efficiently mapping tasks to VMs is needed to maintain dynamic and heterogeneous characteristics of cloud computing environments. This problem has several extra factors which make it even more difficult to deal with (1) the various QoS requirements of cloud users, such as the service cost and the response time, (2) the different types of the combination of the cloud service for
tasks execution, and (3) a large amount of data transmission. The resulting task scheduling problem can be seen as a combinational problem, whose global optima cannot be easily obtained or approximated by traditional algorithms. In fact, it has proved to be an NP-complete problem [8] which cannot be solved in a polynomial time.

Task scheduling for cloud computing systems has been widely studied in a number of previous studies [9–13]. However, after analyzing a large number of task scheduling strategies, we observe that most of the studies have focused on only a few QoS parameters, such as the response time and makespan. Furthermore, there is a lack of appropriate cost models [14, 15] hindering the current scheduling algorithms from analyzing realistic situations. Additionally, scheduling strategies do not reflect the realistic scenarios; most of them consider the pool of cloud resource in the experiments as a fixed one, and few of them optimize the cost and execution time by applying the public cloud pricing model simultaneously. The cost and execution time can influence the tasks greatly, for instance, energy consumption and the reliability may be directly affected by the execution time, and the pay-as-you-go model of cloud providers will be presented through cost.

In this paper, an improved coral reef optimization (ICRO) algorithm is proposed to address this task scheduling problem. The ICRO evacuates configurations with different number of resources, while minimizing the cost of the scheduling, given by the Amazon EC2 cloud pricing model [6], and also the makespan of the scheduling.

The main contributions of this work are the following:

(i) We establish a resource scheduling model over a highly complex cloud environment.

(ii) We propose an approach to solve the problem based on the ICRO algorithm, considering the optimization of makespan and cost of the scheduling.

(iii) We propose new recombination operators to enhance the load balance among virtual machines and to adjust the number of resource users required.

(iv) We test the effectiveness of the proposed scheduling approach on different scenarios, with different performances and metrics to demonstrate the approach’s robustness and convergence. We also compare it with alternative existing strategies through experimental simulation.

The rest of the paper has been structured as follows. Section 2 discusses related works in the literature. Section 3 describes the system model and problem formulation. In Section 4, the detail of the proposed scheduling approach is presented. Extensive experiments are conducted to evaluate the efficacy of our design in Section 5. Section 6 concludes the paper with some final conclusions and remarks.

2. Related Work

Resource and task scheduling [16–19] is a very important part of heterogeneous cloud computing systems because it considerably influences the quality of services, profit, and cost during the operation process of the system. There are several resources in cloud computing with different characteristics such as processors, storage, memory, and software, among others. In turn, the huge amount of tasks submitted by the users need to be handled with these resources. This makes the resource and task scheduling a serious challenge for cloud computing providers.

In recent years, metaheuristic algorithms have been employed to deal with this resource scheduling problem instead of the heuristic algorithms which can give approximate optimal solutions in polynomial time. The heuristic algorithms proposed in the literature for this problem mainly include three different types of approaches: list scheduling, clustering, and duplication-based scheduling [20]. In list scheduling, there are two main phases in a cloud computing resource scheduling process: in the first phase, priorities should be assigned to each subtask that will proceed in order. The subtask will be assigned to processors in the second phase. Priorities are usually determined by the communication and computational cost, such as the heterogeneous earliest finish time (HEFT) [15] or the critical path on a processor (CPOP) [15]. In clustering algorithms, collections of tasks named clusters will be assigned to the appropriate processor, such as MLA [21]. Some duplication-based algorithms exist [22]. They consider duplicate tasks to minimize the makespan. Note that, in general, deterministic heuristic cannot produce good quality nor consistent results when the size of the problems grows.

Differently from the heuristic-based algorithms, the metaheuristic approach can obtain competitive solutions even in very hard problems, by including different search processes with probabilistic characteristics [23]. A few metaheuristic scheduling algorithms have proposed for effectively allocating cloud resources from the provider viewpoint of provider efficiency, such as load balance, utilization maximization, and energy consumption. In [24], a Bee colony-based balancing algorithm was proposed, which can balance the work load among VMs by considering the priorities of tasks that have been moved from heavily loaded VMs. Although this approach can reduce the response time and balance the load, it does not take into consideration the total cost during the process. In [25], Mohammed et al. proposed a discrete symbiotic organism search optimization algorithm for task scheduling problem. This scheduling algorithm aims to reduce the makespan and minimize the degree of imbalance among VMs. However, it does not employ any strategy to enhance the balance among VMs. Additionally, it does not take into account the cost during the scheduling process. In [26], Wen et al. proposed a hybrid algorithm consists of ACO and PSO to improve the cloud resources and utilization radio. In this algorithm, the PSO process helped to avoid ACO premature. In [27], a shadow price strategy was proposed to guide the evolution direction of genetic algorithm, which can reflect the energy consumption of cloud resources. The results show that this algorithm is better than the traditional GA-based scheduling algorithm.

In addition to provider efficiency-oriented cloud resource scheduling, there also exist numerous works on scheduling algorithms from user’s point of view, such as the
makespan, user cost, and application performance. In [28], the authors delivered a cost efficient scheduling algorithm for parallel task scheduling. To reduce competition for low-cost tasks, this method uses a weighted random schedule policy to assign the resources to tasks. Nonetheless, they introduced a fixed number of resources in the algorithm without considering the variety of resources. In [29], the authors produced a hybrid bicriteria scheduling algorithm to schedule the workflow in cloud. This approach aims to minimize the criteria that are chosen and specified by users. Experimental results proved that this approach surpassed the join shortest queue-scheduling algorithm. Although this approach uses the number of resources, which can be specified by the user, it cannot guarantee the optimal utilization of the cloud computing system. Zuo et al. [30] introduced an approach to address the task scheduling problem. In their algorithm, a resource allocation framework was established; service providers were allowed to outsource their tasks to an external cloud. A self-adaptive learning particle swarm optimization (PSO) scheduling algorithm was employed to allocate users’ tasks to maximize the profit of the service provider. However, this work in [24] does not provide an appropriate method to set the number of VMs or control the number of resources. In [31], a communication and storage-aware multiobjective scheduling strategy was employed to solve the scheduling problem of large sets of independent bag of tasks. A cooperative game theoretic-based algorithm was introduced for task scheduling. Although this scheduler can optimize the cost and makespan while considering bandwidth constraints in different VMs, it has no strategy to select the proper number of VMs. Furthermore, it does not consider the load balancing among VMs.

In [32], Bahman et al. presented an improved genetic algorithm-based scheduling approach to achieve the optimal makespan. In this approach, a behavioral modeling approach based on model checking technique was proposed to analyze the correctness, and a Labeled Transition System (LTS) method was used to get the best performance in verification of the proposed algorithm. However, this approach considers neither the cost optimization nor the different numbers of VMs.

Zhang et al. [13] proposed a scheduling scheme based on an ordinal optimization (OO) method. The scheduler applied the workflow on a constant number of VMs. In order to reduce the scheduling time, they extended the OO method originally developed for the complex dynamic system to reduce the search space. Experimental results proved that this method could significantly reduce the scheduling overhead time compared with the Monte Carlo based on repetitive random sampling. Nevertheless, this scheme lacks the guidance for the user to select the number of resources.

In [33], the authors introduced an adaptive scheduling algorithm for scientific workflow in the cloud. Besides, scheduling the workflow on three objectives, makespan, reliability, and final cost, this approach can also scale up and down the resources based on provenance data captured and queried at runtime. Experimental results show that this approach outperforms the MapReduce system [34].

2.1. Coral Reef Optimization. Coral reef optimization (CRO) is a metaheuristic algorithm, based on coral’s reproduction and reefs’ formulation [35], proposed in [36] and has been applied to various complex optimization problem [37–39]. CRO presents very good properties of convergence to global optima. In [40], the CRO has been employed to a specific problem of turbines’ layout in the case of offshore wind farm, which showed advantages against other metaheuristic algorithms, such as differential evolution, harmony search algorithm, and specially evolutionary approaches. In [41], the CRO-SL algorithm has been successfully employed to solve the problem of battery scheduling in a microgrid with renewable generation and variable prices of electricity, which can clearly improve the deterministic use of the battery in all the tested cases. In [42], the CRO has been applied to the optimal parameters’ identification of a permanent magnet synchronous motor, which outperforms a particle swarm optimization algorithm with least squares in obtaining the best set of parameters for the motor. Our proposed ICRO with improved multicrossover and load balance-aware mutation is much more powerful than the original CRO version, since it includes different procedures for doing the search.

From this analysis of the exiting literature, we can observe that the CRO has been successfully applied to different optimization problems. On the contrary, research on the field of the cloud resource scheduling has achieved some results, but there are still some shortcomings, for instance, the user’s QoS has not been effectively guaranteed and the load balance among virtual machines is not fully considered at the same time. To tackle these issues, we propose here an improved version of the CRO, the ICRO-based scheduling algorithm, which not only takes into account how to effectively reduce the cost and makespan but also considers the load balance among virtual machines.

3. Modeling and Formulation

This section presents the system framework and model of the scheduling problem.

3.1. System Framework. The system framework of resource allocation is shown in Figure 1. In this framework, users’ requests are submitted to the system through the user request analyzer, and then, after analyzing, these requests are forwarded to the request manager. The request manager takes responsibility for collecting and managing all the accepted requests. The global and local resource monitors work together to manage the cloud resources, i.e., VMs. The local monitor works for the local resource nodes. The scheduler is the key component and schedules the task to the cloud resources employing the proposed scheduling method. Specifically, first, the scheduler collects the requests’ and resources’ information from the request manager and global resource monitor, respectively. Then, it analyzes the requirements of the tasks and the situation of resources. Finally, it allocates the proper resources to the tasks.
3.2. Problem Modeling. Our work focuses on parallel and independent tasks. Each task may be massive-scale data processing, scientific computing, and large-scale image rendering. For applications such as data mining or sensor data analysis [43], this type of task is usually applied.

Let $VM = \{VM_1, VM_2, \ldots, VM_m\}$ be a set of virtual machines, and let $T = \{T_1, T_2, \ldots, T_n\}$ be a set of tasks. Each task must be executed on just one VM. The task cannot be interrupted once they begin to run. In this paper, we aim at allocating the set of tasks to cloud resources to minimize the cost and makespan of the assignment. As the main constraint of the problem, note that, in any time slot $s (s \in 1, 2, \ldots, S)$, the total resources employed by all tasks must not exceed the total resources offered by providers.

To formulate this problem, an illustration of the parameter definition is presented in Tables 1 and 2. This problem is formulated as the following model.

Minimize

$$\text{makespan} = \max_{j=1}^{m} \sum_{i=1}^{n} y_{ij} \cdot et_{ij}, \quad (1)$$

$$\text{Cost} = \sum_{j=1}^{m} \sum_{i=1}^{n} y_{ij} \cdot c_{ij} \cdot et_{ij}, \quad (2)$$

which subject to

$$\sum_{i=1}^{n} \sum_{j=1}^{m} z_{ij} \cdot \text{cpu}_{ij} \leq \text{cpu}_{\text{total}}, \quad \forall s \in (1, 2, \ldots, S), \quad (3)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} z_{ij} \cdot \text{mem}_{ij} \leq \text{mem}_{\text{total}}, \quad \forall s \in (1, 2, \ldots, S), \quad (4)$$

$$\sum_{j=1}^{n} y_{ij} = 1, \quad T_i \in T, \quad (5)$$

$$y_{ij} \in [0, 1], \quad \forall i \in \{1, 2, \ldots, n\}, \quad j \in \{1, 2, \ldots, m\}. \quad (6)$$

Equation (1) represents the makespan of the tasks submitted by users. Equation (2) shows the total cost of all tasks executed on the VMs. Constraints (3) and (4) guarantee that the provider cannot use CPU and memory resources beyond its capacity. Constraints (5) and (6) ensure that each task must be executed and only be executed on one VM.

4. Proposed Scheduling Approach

In this paper, we propose an improved CRO-based scheduling approach to address the aforementioned scheduling problem. In the ICRO, each coral (potential solution to the problem) represents a set of resources to which tasks should be allocated. It experiences the process of sexual reproduction (broadcast spawning and brooding), asexual reproduction (budding), and depredation. Finally, the healthier corals will survive, which means that the best solutions obtained by the algorithm will be retained and promoted in the next steps of the ICRO. To enhance the performance of the original CRO for this scheduling problem, a multicrossover, load balance-aware mutation, and better-offspring strategies are applied.

4.1. Original CRO. In the original CRO [35], each coral (representing a candidate solution of the problem) can be allocated on a square of the reef which owns M-N square grids. The CRO begins with randomly assigning some squares in the reef to be occupied by corals, but some of the others will be empty. The percentage of the occupied squares in the reef is very important for the algorithm’s dynamics and evolution, and it is usually set as $\rho = 0.8$. Each coral will be assessed by a health function, that is to say, the healthier the corals are, the most probable they will survive for the next generations of the algorithm.

After initialization, CRO carries out the second phase of the reef formation. In this phase, CRO simulates the corals’ reproduction by applying a series of operations: sexual reproduction (broadcast spawning and brooding), asexual reproduction (budding) [35], and polyps depredation. After the sexual and asexual reproduction, a set of larvae is formed [35], in order to find a square to grow, and they will try to fight against the other corals that have been allocated in that square or get the square freely if it is empty. If larvae fail to be allocated a place to grow, they will be perished. This process will be repeated until a given stop criterion is met. The steps of the original CRO are summarized as follows:

Step 1: randomly assign some squares of the reef to be occupied by corals.

Step 2: broadcast spawning (external sexual reproduction). This step includes two steps:

Step 2.1: randomly select a fraction of corals in the reef to broadcast spawners.

Step 2.2: randomly select couples from the spawners, each of which will produce coral larvae by the crossover operation, which will be released out into the water.

Step 3 (brooding (internal sexual reproduction)): in this step, unselected corals will form coral larvae by randomly mutating the brooding-reproductive corals; then, coral larvae will be released out into the water.

Step 4 (larvae setting): healthiness (fitness) of each formed coral larva will be computed by the health function, and then, the coral larvae that owns the better health function will be allocated to the square that has been occupied by the existing coral.

Step 5 (asexual reproduction): the existing corals in the reef will be sorted by the healthiness, and a fraction of them will duplicate themselves and try to settle in different places of the reef by following the process described in Step 4.

Step 6 (depredation in polyp phase): a small number of corals in the reef will be depredated at the end of the reproduction in order to release the space in the reef for the next coral generation.
4.2. ICRO-Based Scheduling Approach. ICRO is to allocate the tasks to cloud resources to optimize the cost and execution time. In this paper, the CRO’s operators are carefully adjusted to distribute tasks on VMs. Firstly, in Step 1, the corals are randomly initialized and assigned to the squares of the reef. Steps 2–11 develop the main loop. Steps 3–6 apply broadcast spawning (external sexual reproduction) operation. The multicrossover operation and the better-offspring strategy are employed to help ICRO to produce better offspring. The proposed mutation operation broods coral larvae in Step 7 (internal sexual reproduction). Asexual reproduction is the same as the original CRO. Step 8 evaluates every coral using equation (7). Afterward, some corals are depredated in the reef to release the space for the next generation. The algorithm stops until the stopping criterion is satisfied and then returns the solution with the best fitness value (Step 12). The pseudocode of ICRO is shown in Algorithm 1.

4.2.1. Coral and Fitness Function. The coral (individual) in the coral population represents a complete task scheduling where each dimension of the coral represents a task and the required VM. The length of the coral is the size of the submitted tasks. Table 3 shows an example of the coral. In it, the position of dimension 1 represents task 1 and the position of dimension 2 represents task 2. Similarly, the value of the first dimensions 1 and 3 shows that task 1 will be assigned to VM3 for execution. Value of dimensions 2 and 4 assigns task 2 to VM4 and so on.

The quality of a coral can be evaluated by the fitness function. Function (7) assigns a fitness value to each coral based on its cost and makespan every generation. The fitness function is shown as follows:

\[
\text{Fit} = \alpha \frac{\text{Cost}_i - \text{Cost}_{\text{min}}}{\text{Cost}_{\text{max}} - \text{Cost}_{\text{min}}} + \beta \frac{\text{Mksp}_i - \text{Mksp}_{\text{min}}}{\text{Mksp}_{\text{max}} - \text{Mksp}_{\text{min}}},
\]

where Cost, is the cost of the i-th coral, Cost_{max} and Cost_{min} represent the maximum and minimum cost of all corals, Mksp_i is the makespan of the i-th coral, Mksp_{max} and Mksp_{min} represent the maximum and minimum makespan of all corals, respectively, and \( \alpha \) and \( \beta \) are makespan and cost weights, respectively, which enable the user to assign the priority to the given optimization objective and \( \alpha + \beta = 1 \). In this paper, they are both set to 0.5.

4.2.2. Broadcast Spawning. The ICRO makes use of the roulette wheel for selection of the corals in the reef to be broadcast spawners and then produces the coral larvae by applying multicrossover operation. In order to enhance the performance and coverage rate of the original CRO, the better-offspring strategy is used.

The multicrossover operation is shown in Figure 2. The three parents’ crossover operation is used in the original CRO. Unlike the original CRO, the multicrossover operation is designed to help CRO to search in a wider range, thereby enhancing the diversity of the population. The detail of the multicrossover operation is presented in Algorithm 2.

The better-offspring strategy is designed to accelerate convergence and therefore enhance the performance of the CRO. It is described in Algorithm 3. Round is the improvement times when applying this strategy. Random mutation is used in this strategy.

4.2.3. Brooding. In ICRO, the brooding (internal sexual reproduction) is similar to the original CRO, except that the load balance-aware mutation operation is employed to reduce the imbalance among VMs. The detail of the proposed mutation operation is shown in Algorithm 4.

The operation of asexual reproduction and depredation remain the same as the original CRO, details of which have been presented in Section 4.

5. Experimental Result and Analysis

5.1. Experiment Design. Several problem instances presented in Table 4 are used to verify the effectiveness of our specific scheduling approach in this work. VM instance types and EC price [6] are set according to [6], which are given in Table 4. The CPU and memory are chosen for this work because there are two typical configurations for users and service providers in cloud [44]. The other parameters of VMs are set to 4 GB RAM, 100 M/s bind width, and 10 GB storage. Three different types of task instances are designed; small size, medium size, and large size of tasks are included in these types of test instances, respectively. These types of test instances are shown in Tables 5–7.

5.2. Methods Compared. The ICRO-based scheduling algorithm is compared with the original CRO-based scheduling algorithm (OCRO-SA), and the standard GA-based scheduling algorithm (GA-SA) [45] and the first-come-first-serve (FCFS) [46] scheduling algorithm.

The parameters of ICRO-SA and OCRO-SA remain the same as those in [36], except that the single-point crossover and the random mutation operation are used in OCRO-SA. These two operations are also employed in GA-SA. The population size is set to 100; the max generation is set to 10,000. For all test instances, the termination criterion of OCRO-SA, ICRO-SA, and GA-SA is to reach the maximum number of generations.

5.3. Experiment Results. The performance of the proposed scheduling approach is evaluated using a cloud simulator based on CloudSim [47–49]. CloudSim is a generalized simulation framework that allows modeling, simulating, and experimenting the cloud computing infrastructure and application services [47]. All simulations run on a PC with 64 bit Intel core i5 CPU and 12 GB memory using windows 10 operation system. Each algorithm is executed 30 independent runs for every test instance. A two-sided Mann–Whitney U test was carried out for each experiment to see
if the results were statistically and significantly different (within a 95% confidence interval) from the corresponding ICRO-SA results. Results in bold are not statistically significant.

5.3.1. Performance on Different Types of Tasks. Makespan and cost are the common indicators for evaluating a scheduling approach. In this section, the average cost and makespan of three different test instances obtained in the 30 runs of all scheduling approaches with 100, 300, 500, and 1000 tasks and 100 VMs are given in Tables 8–10 and Figures 3 and 4.

We can observe that the average makespan and cost obtained by ICRO-SA is better than that of the compared approaches in almost all the test cases. For instance, ICRO-SA improves the makespan by 77.08%, 90.25%, and 636.12% for the large size task instance (1000 tasks) compared with other three approaches, respectively, and improves the cost for the medium size task instance (300 tasks) by 6.58%, 9.28%, and 40.99%, respectively. This is because ICRO has a better search ability due to its multicrossover operation and better-offspring strategies, which can maintain diversity and speed up the algorithm’s convergence during the scheduling process.

Results, obtained by GA-SA are also better than those of OCRO-SA, which implies that, by improving the operations of the original CRO, the proposed scheduling approach ICRO-SA can better deal with the cloud scheduling problem. It also can be seen that the proposed approach achieves more performance improvement for the relatively complex test instance (large-size task instance) than for the small size task instance, which means that ICRO-SA is more capable of handling the much more complex task scheduling problem than the compared approaches. where \( T_{E_{\text{avg}}} \) is the average execution time of tasks on the \( i \)th VM, \( T_{E_{\text{avg}}} \) is the average execution time among all VMs, and \( m \) is the number of VMs.

5.3.2. Algorithm Convergence and Load Balance. This section analyzes the global exploration ability of ICRO-SA. The convergence curve of OCRO-SA, ICRO-SA, and GA-SA for three test instances is shown in Figures 5–7, respectively. It can be observed that, in the early search stage, the original CRO-SA can converge more quickly than ICRO but cannot make any improvement in later evolutions (in Figure 5, after 1600 evolutions), while ICRO has the continuous exploration ability to find a better solution, which can be explained by the fact that the better-offspring strategy can generate a better offspring in each iteration to speed up the search.

| Notations | Definition |
|-----------|------------|
| \( n \) | Number of tasks |
| \( m \) | Number of VMs |
| \( c_j \) | Cost of the \( j \)th VM |
| \( d_i \) | Deadline of the \( i \)th tasks |
| \( et_{ij} \) | Execute time of the \( i \)th task on the \( j \)th VM |
| \( ft_{ij} \) | The finish time of the \( i \)th task on \( j \)th VM |
| \( cpu_j \) | Number of CPU resource for the \( j \)th VM |
| \( mem_j \) | Size of memory for the \( j \)th VM |
| \( mem_{\text{total}} \) | Total size of memory |
| \( cpu_{\text{total}} \) | Total number of CPU resource |

Figure 1: The system framework model.
process. GA-SA has the lowest convergence speed compared with the other approaches. We can also observe that, in Figure 5, OCRO-SA and ICRO-SA almost have the same convergence performance due to which the number of tasks is 100, and the number of VMs is also 100; thus, assigning these tasks to VMs becomes easier.

**Table 2: Decision variables.**

| Notation | Definition                                                                 |
|----------|-----------------------------------------------------------------------------|
| $y_{ij}$ | Binary decision variable $y_{ij} = 1$ such that if the $i$th task is allocated to the $j$th VM; otherwise, $y_{ij} = 0$ |
| $z_{ij}$ | Binary decision variable such that $z_{ij} = 1$ if the $i$th task is allocated to the $j$th VM in the time slot $s$ |

**Algorithm 1: ICRO.**

```
Require: task and VM set  
Ensure: scheduling plan  
(1) Initialize corals and assign them to reef randomly;  
(2) while the stopping criterion is not met do  
(3) Broadcast spawners;  
(4) -Select corals to be the broadcast spawners;  
(5) -Produce coral larvae by the proposed crossover operation;  
(6) -Mutate the offspring to find a better one;  
(7) Brood larvae by the proposed mutation operation;  
(8) Compute the healthiness function (fitness function);  
(9) Asexual Reproduction;  
(10) Depredation;  
(11) end while  
(12) Return the best solution;  
```

**Figure 2: Multicrossover operation.**

**Table 3: Coral encoding.**

| Dimension | Index of VM |
|-----------|-------------|
| 1         | 3           |
| 2         | 4           |
| 3         | 6           |
| 4         | 2           |
| 5         | 3           |

**Algorithm 2: Multicrossover operation.**

```
Require: corals  
Ensure: offspring  
(1) Randomly select $k$ corals from the coral population as the parents.  
(2) Generate $d^*k$ random values for each dimension of all corals. $d$ is the size of the dimension. The random value $v(i, j)$ is ranged from $0$ to $1$ and used for the $i$th dimension of the $j$th parent.  
(3) Set the $i$th dimension of the offspring to be the same as the corresponding dimension of the $j$th parent if $v(i, j)$ is the largest among the parents. This process will continue until all the dimensions have been considered.  
(4) Return the offspring.  
```
Require: coral, round
Ensure: offspring
(1) Let $i = 0$;
(2) while $d < \text{dimension}$ do
(3) Apply mutation on the input coral;
(4) Compute the fitness of the mutated coral;
(5) Compare the fitness of the mutated coral with the input coral; then, record the better one in the offspring;
(6) Increment $i$;
(7) end while
(8) return the Offspring;

**Algorithm 3**: Better-offspring strategy.

Require: corals
Ensure: mutated coral
(1) Let $d = 0$;
(2) while $d < \text{dimension}$ do
(3) Randomly generate a value from 0 to 1;
(4) if the random value $< \text{mutate probability}$ then
(5) Calculate the total number of tasks on each VM according to the information of the coral;
(6) Find $VM_{\text{min}}$, on which the minimum number of tasks are executed;
(7) Set the $d$th dimension of the coral to be the index of $VM_{\text{min}}$;
(8) end if
(9) Increment $d$;
(10) end while
(11)
(12) return the Offspring;

**Algorithm 4**: Load balance-aware mutation.

**Table 4**: VM types and price.

| Type      | CPU | Memory | Price |
|-----------|-----|--------|-------|
| t2.small  | 1   | 2      | 0.03  |
| t2.medium | 2   | 4      | 0.06  |
| t2.large  | 2   | 8      | 0.12  |
| t2.xlarge | 4   | 16     | 0.24  |
| t2.2xlarge| 8   | 32     | 0.48  |
| m3.medium | 1   | 3.75   | 0.098 |
| m3.large  | 2   | 7.5    | 0.196 |
| m3.xlarge | 4   | 15     | 0.392 |

**Table 5**: Small-size tasks.

| Parameter       | Value            |
|-----------------|------------------|
| Length (CPU)    | [500, 20000] MIPS|
| File size       | [200, 1000] MB   |
| Out size (memory)| [20, 40] MB      |

**Table 6**: Medium-size tasks.

| Parameter       | Value            |
|-----------------|------------------|
| Length (CPU)    | [10000, 200000] MIPS|
| File size       | [400, 1500] MB   |
| Out size (memory)| [30, 50] MB      |
The load balance is very important in some cases where a large number of tasks need to be assigned to the limited cloud resources. It can also reflect the algorithm’s performance on cloud computing scheduling. We use the metric of the degree of imbalance (DI) to illustrate the load balance, which is calculated as follows.

\[
    DI = \sqrt{\frac{\sum_{i=1}^{m} (TE_i - TE_{\text{avg}})^2}{m}},
\]

Table 7: Large-size tasks.

| Parameter              | Value                  |
|------------------------|------------------------|
| Length (CPU)           | [100000, 1000000] MIPS |
| File size              | [800, 2000] MB         |
| Out size (memory)      | [40, 60] MB            |

The DI results obtained by the four scheduling algorithms for the test instances are given in Table 11. It can be observed that the DI results obtained by ICRO-SA are better than almost all the other compared scheduling approaches in the three test instances. It means that, on the one hand, the proposed algorithm can obtain shorter makespan than the
other compared approaches with an acceptable balanced workload; meanwhile, it has a lower cost. On the other hand, in order to achieve a better makespan and cost, the load balance will be sacrificed. The execution time of tasks on each VM is shown in Figures 8–10. For the 22th, 48th, and 69th VM, with the similar CPU power in Figure 8, however,
the 22th and 48th VM is much cheaper than 69th VM; the
deal scheduling is not assigning more tasks on the 69th VM.
In fact, ICRO-SA does not assign tasks to the 69th VM, but
assigns tasks to other cheaper VMs. Similar to the 12th, 37th,
and 57th VM, with the similar CPU power in Figure 9, the
ICRO-SA only assigns tasks to the 101th VM, which has the
lower price than the 33th and 55th VM.

5.3.3. Impact of the Number of VMs. In this section, ICRO-
SA’s behavior is analyzed and compared with the other
algorithms. Four scheduling approaches will produce
scheduling plans for all possible VMs and will start
scheduling from one VM, and then, increase the number of
VMs until the stop criterion is met. These results are pre-
sented in Figures 11 and 12.

From these results, we can observe that the scheduling
approach does not apply more cloud resources beyond a
particular number of VMs. For instance, ICRO-SA achieves
a value (4.85) with 150 VMs, as shown in Figure 11; nev-
bertheless, it cannot be improved when incrementing the
number of VMs. It can also be observed that ICRO-SA can
obtain very similar execution time while employing a lower
number of VMs. When the makespan is 150, the number of
VMs employed by ICRO-SA, OCRO-SA, and GA-SA is 60,
80, and 80, respectively. In general, all four approaches in
this paper do not need a large number of VMs to get the
minimum makespan and cost.
Figure 7: Average evolutionary curves for the test instance of large-size task (100 tasks).

Figure 8: Execution time of each VM for the test instance of small-size task (1000 tasks).

Figure 9: Execution time of each VM for the test instance of medium-size task (500 tasks).
Figure 10: Execution time of each VM for the test instance of large-size task (100 tasks).

Table 11: Degree of imbalance for all test distances.

| Type    | Task   | ICRO-SA   | OCRO-SA   | GA-SA    | FCFS-SA   |
|---------|--------|-----------|-----------|----------|-----------|
| Small   | 100    | 0.8128    | 0.8966    | 0.8652   | 3.8994    |
|         | 300    | 2.2846    | 2.6681    | 2.3767   | 11.6983   |
|         | 500    | 3.3332    | 4.3217    | 4.2271   | 19.4971   |
|         | 1000   | 585.95%   | 1036.84%  | 1107.52% | 38.9942   |
| Medium  | 100    | 9.2122    | 9.5655    | 9.7713   | 40.7169   |
|         | 300    | 2675.95%  | 2831.74%  | 2599.19% | 122.1507  |
|         | 500    | 43.5723   | 44.4196   | 45.3369  | 203.3845  |
|         | 1000   | 7132.38%  | 10700.78% | 11031.15%| 407.169   |
| Large   | 100    | 55.6169   | 53.3243   | 53.7859  | 220.7845  |
|         | 300    | 15140.36% | 15814.60% | 13758.05%| 662.3534  |
|         | 500    | 225.9604  | 260.198   | 237.8428 | 1103.9224 |
|         | 1000   | 38987.90% | 54557.86% | 59925.31%| 2207.8447 |

Figure 11: Execution time results with a different number of VMs for the test instance of small-size tasks (300 tasks).
6. Conclusion

In this paper, we have presented a resource scheduling system for the resource allocation problem in the cloud computing environment. An improved Coral Reef Optimization-based (ICRO) scheduling approach has been proposed for this problem. ICRO enhances the original CRO through purposeful modifications in its internal and external sexual reproducing operations. The proposed ICRO implements a multicrossover and better-offspring strategies to produce more fit offspring rather than the original reproduction operation. It also employs a balance-aware mutation to enhance the load balance among virtual machines. These improvements can decrease the randomness of the algorithm and focus the search, compared to the original CRO. Experimental results show that the proposed approach is effective and efficient for this scheduling problem. ICRO obtained results are better than the original CRO and also better than those obtained by the standard GA and FCFS for different types of tasks, under a reasonable runtime.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the Engineering and Physical Sciences Research Council of UK, under Grant EP/K001310/1, Hubei Key Laboratory of Intelligent Vision Based Monitoring for Hydroelectric Engineering, China Three Gorges University, China, under Grant no. 2017SDSJ06, and National Natural Science Foundation of China, under Grant no. U1703261.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility,” Future Generation Computer Systems, vol. 25, no. 6, pp. 599–616, 2009.
[2] M. H. Ghahramani, M. Zhou, and C. T. Hon, “Toward cloud computing QoS architecture: analysis of cloud systems and cloud services,” IEEE/CAA Journal of Automtica Sinica, vol. 4, no. 1, pp. 6–18, 2017.
[3] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the clouds,” ACM SIGCOMM Computer Communication Review, vol. 39, no. 1, pp. 50–55, 2008.
[4] S. Bhardwaj, L. Jain, and S. Jain, “Cloud computing: a study of infrastructure as a service (IaaS),” International Journal of Engineering and Information Technology, vol. 2, no. 1, pp. 60–63, 2010.
[5] Z. Zhu, G. Zhang, M. Li, and X. Liu, “Evolutionary multi-objective workflow scheduling in cloud,” IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 5, pp. 1344–1357, 2016.
[6] “Amazon Ec2,” 2017. [Online]. Available:http://aws.amazon.com/ec2.
[7] S. Yassa, R. Chelouah, H. Kadima, and B. Granado, “Multi-objective approach for energy-aware workflow scheduling in cloud computing environments,” The Scientific World Journal, vol. 2013, Article ID 350934, 13 pages, 2013.
[8] J. D. Ullman, “Np-complete scheduling problems,” Journal of Computer and System Sciences, vol. 10, no. 3, pp. 384–393, 1975.
[9] Y. Laili, F. Tao, L. Zhang, Y. Cheng, Y. Luo, and B. R. Sarker, “A ranking chaos algorithm for dual scheduling of cloud service and computing resource in private cloud,” Computers in Industry, vol. 64, no. 4, pp. 448–463, 2013.
[10] L. Grandinetti, O. Piscane, and M. Sheikhalishahi, “An approximate -constraint method for a multi-objective job scheduling in the cloud,” Future Generation Computer Systems, vol. 29, no. 8, pp. 1901–1908, 2013.
[11] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, “Cloud computing resource scheduling and a survey of its evolutionary approaches,” ACM Computing Surveys (CSUR), vol. 47, no. 4, p. 63, 2015.

[12] T. Kaur and I. Chana, “Energy aware scheduling of deadline-constrained tasks in cloud computing,” Cluster Computing, vol. 19, no. 2, pp. 679–698, 2016.

[13] F. Zhang, J. Cao, K. Li, S. U. Khan, and K. Hwang, “Multi-objective scheduling of many tasks in cloud platforms,” Future Generation Computer Systems, vol. 37, pp. 309–320, 2014.

[14] L. Deng, Q. Yu, and J. Peng, “Adaptive scheduling strategies for cloud-based resource infrastructures,” Security and Communication Networks, vol. 5, no. 10, pp. 1102–1111, 2012.

[15] H. Topcuoglu, S. Hariri, and M.-y. Min-You Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260–274, 2002.

[16] P. Zhang and M. Zhou, “Dynamic cloud task scheduling based on a two-stage strategy,” IEEE Transactions on Automation Science and Engineering, vol. 15, no. 2, pp. 772–783, 2018.

[17] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, “A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems,” IEEE/CAA Journal of Automatica Sinica, vol. 6, no. 4, pp. 904–916, 2019.

[18] H. Yuan, J. Bi, W. Tan, and B. H. Li, “Temporal task scheduling with constrained service delay for profit maximization in hybrid clouds,” IEEE Transactions on Automation Science and Engineering, vol. 14, no. 1, pp. 337–348, 2017.

[19] H. Yuan, M. Zhou, Q. Liu, and A. Abusorrah, “Fine-grained resource provisioning and task scheduling for heterogeneous applications in distributed green clouds,” IEEE/CAA Journal of Automatica Sinica, vol. 7, no. 5, pp. 1380–1393, 2020.

[20] M. A. Khan, “Scheduling for heterogeneous systems using constrained critical paths,” Parallel Computing, vol. 38, no. 4-5, pp. 175–193, 2012.

[21] B. Uçar, C. Aykanat, K. Kaya, and M. Ikcini, “Task assignment in heterogeneous computing systems,” Journal Of Parallel And Distributed Computing, vol. 66, no. 1, pp. 32–46, 2006.

[22] O. Sinnen, A. To, and M. Kaur, “Contention-aware scheduling with task duplication,” Journal of Parallel and Distributed Computing, vol. 71, no. 1, pp. 77–86, 2011.

[23] S. Gupta, V. Kumar, and G. Agarwal, “Task scheduling in multiprocessor system using genetic algorithm,” in Proceedings of the 2010 Second International Conference on Machine Learning and Computing (ICMLC), pp. 267–271, IEEE, Bangalore, India, February 2010.

[24] D. B. Venkata Krishna and P. V. Krishna, “Honey bee behavior inspired load balancing of tasks in cloud computing environments,” Applied Soft Computing, vol. 13, no. 5, pp. 2292–2303, 2013.

[25] M. M. Abdullahi, M. A. Ngadi, and S. I. M. Abdulhamid, “Symbiotic organism search optimization based task scheduling in cloud computing environment,” Future Generation Computer Systems, vol. 56, pp. 640–650, 2016.

[26] X. Wen, M. Huang, and J. Shi, “Study on resources scheduling based on aco all algorithm and PSO algorithm in cloud computing,” in Proceedings of the International Symposium On Distributed Computing And Applications to Business, Engineering & Science, pp. 219–222, Wuhan, China, November 2012.

[27] G. Shen and Y.-Q. Zhang, A Shadow Price Guided Genetic Algorithm for Energy Aware Task Scheduling on Cloud Computers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[28] Y. Xin, Z.-Q. Xie, and J. Yang, “A load balance oriented cost efficient scheduling method for parallel tasks,” Journal of Network and Computer Applications, vol. 81, pp. 37–46, 2017.

[29] H. Klob, B. Schulze, R. Pinto, and A. Mury, “A bi-criteria scheduling process with CoS support on grids and clouds,” Concurrency and Computation: Practice and Experience, vol. 24, no. 13, pp. 1443–1460, 2012.

[30] X. Zuo, G. Zhang, and W. Tan, “Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud,” IEEE Transactions on Automation Science and Engineering, vol. 11, no. 2, pp. 564–573, 2014.

[31] R. Rubing Duan, R. Prodan, and X. Xiao-rong Li, “Multi-objective game theoretic scheduling of bag-of-tasks workflows on hybrid clouds,” IEEE Transactions on Cloud Computing, vol. 2, no. 1, pp. 29–42, 2014.

[32] B. Keshanchi, A. Souri, and N. J. Navimipour, “An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing,” Journal of Systems and Software, vol. 124, pp. 1–21, 2017.

[33] L. Carro-Calvo, A. Portilla-Figueras, and L. Prieto, “An offshored cruising optimization problem,” in Proceedings of the 2014 Fourth International Conference on Smart Grid and Clean Energy Systems (SmartGrids), pp. 1–7, 2014.

[34] S. Salcedo-Sanz, C. Casanova-Mateo, A. Pastor-Sánchez, M. Sánchez-Girón, and S. Jiménez-Fernández, “A novel coral reefs optimization algorithm: a novel metaheuristic for efficiently solving optimization problems,” The Scientific World Journal, vol. 2014, Article ID 739768, 15 pages, 2014.

[35] S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López, and J. A. Portilla-Figuera, “The coral reefs optimization algorithm: a novel metaheuristic for efficiently solving optimization problems,” in Proceedings of the 15th International Conference On Applied Stochastic Models And Data Analysis (ASMDA2013), pp. 751–758, Mataró (Barcelona), Spain, June 2013.

[36] S. Salcedo-Sanz, C. Casanova-Mateo, A. Pastor-Sánchez, and M. Sánchez-Girón, “A coral reefs optimization algorithm: an efficient meta-heuristic for solving hard optimization problems,” in Proceedings of the 10th International Conference On Applied Stochastic Models And Data Analysis (ASMDA2013), pp. 751–758, Mataró (Barcelona), Spain, June 2013.

[37] S. Salcedo-Sanz, A. Portilla-Figuera, L. Prieto, A. Blanco-Aguilera, and R. Garcia-Herrera, “Feature selection in wind speed prediction systems based on a hybrid coral reefs optimization-extreme learning machine approach,” Renewable Energy, vol. 105, pp. 91–98, 2014.

[38] S. Salcedo-Sanz, J. Del Ser, and S. Gil-López, “A coral reefs optimization algorithm for optimal mobile network deployment with electromagnetic pollution control criterion,” Applied Soft Computing, vol. 14, pp. 239–248, 2014.

[39] S. Salcedo-Sanz, J. Del Ser, and A. Portilla-Figuera, “A coral reefs optimization algorithm for optimal mobile network deployment with electromagnetic pollution control criterion,” Applied Soft Computing, vol. 14, pp. 239–248, 2014.

[40] S. Salcedo-Sanz, P. García-Díaz, J. A. Portilla-Figuera, J. Del Ser, and S. Gil-López, “A coral reefs optimization algorithm for optimal mobile network deployment with electromagnetic pollution control criterion,” Applied Soft Computing, vol. 14, pp. 239–248, 2014.

[41] S. Salcedo-Sanz, C. Camacho-Gómez, R. Mallol-Poyato, S. Jiménez-Fernández, and J. Del Ser, “A novel coral reefs optimization algorithm with substrate layers for optimal battery scheduling optimization in micro-grids,” Soft Computing, vol. 20, no. 11, pp. 4287–4300, 2016.

[42] Y. Quan, T. Na, Z. Ji, and W. Yan, “Coral reefs optimization for solving parameter identification in permanent magnet
synchronous motor,” *Journal of System Simulation*, vol. 28, no. 4, pp. 927–939, 2016.

[43] H. Liu and D. Orban, “Gridbatch: Cloud computing for large-scale data-intensive batch applications,” in *Proceedings of the 8th IEEE International Symposium On Cluster Computing And the Grid*, pp. 295–305, Madrid, Spain, May 2008.

[44] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, “Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads,” in *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*, pp. 228–235, Noida, India, September 2010.

[45] J. H. Holland, “Adaptation in natural and artificial systems,” *An Introductory Analysis with Application to Biology, Control, and Artificial Intelligence*, University of Michigan Press, Ann Arbor, MI, USA, 1975.

[46] U. Schwiegelshohn and R. Yahyapour, “Analysis of first-come-first-serve parallel job scheduling,” *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, vol. 98, pp. 629–638, 1998.

[47] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[48] R. N. Calheiros, R. Ranjan, C. A. De Rose, and R. Buyya, “Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services,” 2009, http://arxiv.org/abs/0903.2525.

[49] R. Buyya, R. Ranjan, and R. N. Calheiros, “Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: challenges and opportunities,” in *Proceedings of the 7th High Performance Computing and Simulation Conference*, pp. 1–11, Dublin, Ireland, July 2009.