A WOA-Based Optimization Approach for Task Scheduling in Cloud Computing Systems

Xuan Chen, Long Cheng *Member, IEEE,* Cong Liu, Qingzhi Liu, Jinwei Liu *Member, IEEE,* Ying Mao *Member, IEEE,* and John Murphy *Senior Member, IEEE*

***Abstract*—Task scheduling in cloud computing can directly affect the resource usage and operational cost of a system. To improve the efficiency of task executions in a cloud, various metaheuristic algorithms, as well as their variations, have been proposed to optimize the scheduling. In this work, for the first time, we apply the latest metaheuristics WOA (the whale optimization algorithm) for cloud task scheduling with a multi-objective optimization model, aiming at improving the performance of a cloud system with given computing resources. On that basis, we propose an advanced approach called IWC (Improved WOA for Cloud task scheduling) to further improve the optimal solution search capability of the WOA-based method. We present the detailed implementation of IWC and our simulation-based experiments show that the proposed IWC has better convergence speed and accuracy in searching for the optimal task scheduling plans, compared to the current metaheuristic algorithms. Moreover, it can also achieve better performance on system resource utilization, in the presence of both small and large-scale tasks.**

***Index Terms*—Cloud computing; task scheduling; whale optimization algorithm; metaheuristics; multi-objective optimization**

I. Introduction

With the ubiquitous growth of Internet access and big data, cloud computing becomes more and more popular in today's business world [1]. Compared to other distributed computing techniques (e.g., cluster and grid computing), cloud computing has provided an elastic and scalable way on delivering services to consumers. Namely, consumers do not need to possess the underlying technology and they can make use of computing resources and platforms in a pay-per-use fashion [2], [3].

The basic mechanism of cloud computing is to dispatch computing tasks to a resource pooling constituting of a large number of heterogeneous virtualized servers or virtual machines (VMs) [4], [5]. As cloud computing is a market-oriented utility, to allow cloud providers and users to maximize their profit and return on investment [6], advanced strategies on resource scheduling, which can support software and user applications, tasks and workflows, etc., are always required. In fact, scheduling can directly affect the performance of a system such as resource usage efficiency and operational cost, and it has been seen as of paramount importance to cloud computing [7].

As VMs can be dynamically provisioned, allocated and managed [8], the scheduling problems in cloud computing can be generally divided into two main layers: the first is the scheduling of the tasks submitted by a user and mapping them to a set of available VM resources; and the second is a VM and host mapping which makes a VM in a suitable host to create or migrate [9]. We focus on optimizing the former problem in this work, because it directly affects the processing capability of a cloud computing system, and an optimized task scheduling will greatly improve the efficiency of the whole system such as the time and price cost [10]. However, the complexity of the optimization problem is NP-hard [11]. This means that the problem solving time will be in exponential time, and an algorithms will suffer from a dimensionality breakdown when the size of the problem grows.

To slove complex optimization problems in an acceptable time, using metaheuristics algorithms has received increasing attention in recent years [12]. The reason is that they are shown to be highly effective and can find approximately optimal solutions in polynomial time rather than exponential time, compared to conventional methods [3], [13]. In fact, various metaheuristics as well as their variations have been used to solve scheduling problems in many fields [14], [15], [16], [17], [18], [19], [20], which also include the cloud computing. As summarized by the latest survey [21], currently metahuristics used in cloud task scheduling mainly include the genetic algorithm (GA) [22] and swarm intelligence algorithms, such as the ant colony optimization (ACO) [23] and the particle swarm optimization (PSO) [24]. These optimization algorithms are derived from the simulations of biological population evolutions, and they can solve complex global optimization problems through cooperation and competition among individuals [25].

The whale optimization algorithm (WOA) is one of the latest metaheuristics [26] that is nature-inspired by the humpback hunting method (i.e., bubble-net predation). Because of this unique optimization mechanism, WOA can provide a good

X. Chen is with Zhejiang Industry Polytechnic College, Zhejiang, China. E-mail: cxuan762@gmail.com

L. Cheng is with the School of Computing, Dublin City University, Ireland. E-mail: long.cheng@dcu.ie (*Corresponding Author*)

C. Liu is with the School of Computer Science and Technology, Shandong University of Technology, China. E-mail: liucongchina@sdust.edu.cn

Q. Liu is with the Information Technology Group, Wageningen University, Netherlands. E-mail: qingzhi.liu@wur.nl

J. Liu is with the Department of Computer and Information Sciences at Florida A&M University, USA. E-mail: jinwei.liu@famu.edu

Y. Mao is with the Department of Computer and Information Science at Fordham University in the New York City. E-mail: ymao41@fordham.edu

J. Murphy is with the School of Computer Science, University College Dublin, Ireland. E-mail: j.murphy@ucd.ie

Part of this work was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 799066, the National Science Foundation of China (61902222), and the Taishan Scholar Youth Program of Shandong Province (tsqn201909109).

global search capability, which makes it become popular in various engineering problems. In this work, we will try to explore the application of the WOA approach to a multi-objective task scheduling optimization problem in cloud computing. Specifically, we focus on optimizing the task execution time, load and price cost of a cloud computing system for given tasks, and these measures will be essential to ensure that the entire configuration of the VMs is as optimal as possible. In general, we first map our task scheduling scheme to the whale foraging model, and thus we can get an approximately optimal solution using the WOA algorithm. On that basis, we propose an advanced approach called IWC (**I**mproved **W**OA for **C**loud task scheduling), which aims to further improve the optimal solution search capability of WOA. We provide the detailed implementation of IWC and conduct a performance evaluation using a large number of simulations with up to 10000 tasks. We summarize the contributions of this work as follows:

- To improve the efficiency of task executions in a cloud computing system, we introduce a multi-objective optimization model for task scheduling and apply the WOA approach to solve the problem.
- We propose a new approach called IWC for more efficient task scheduling by incorporating advanced optimization strategies to improve both the convergence speed and accuracy of the WOA-based approach.
- We present the detailed design and implementation of IWC and compare it with some existing metaheuristics including ACO and PSO. Our experimental results demonstrate that IWC can achieve better performance on system resource utilization for both small and large-scale tasks in cloud computing.

The rest of this paper is organized as follows. In Section II, we report the related work. In Section III, we introduce our task scheduling optimization model. We present the proposed IWC approach and its implementation details in Section IV. We carry out extensive evaluation of our approach in Section V and conclude this paper in Section VI.

II. Related Work

Task scheduling strategies which can efficiently allocate resources to required tasks under constraints are still challenging current cloud computing techniqus. This is because the requirements such as bandwidth, storage, resource expenses, and response time may differ for each task, which makes the optimization problem very complex, and the heterogeneity and dynamicity of the cloud computing environment will also further complicate the problem [4].

In order to efficiently use cloud resources, a lot of mathematical task scheduling solutions have been proposed. For example, Malawski et al. [27] modeled the relationship between the deadline and cost on hybrid clouds as a mixed integer nonlinear programming problem with an implementation in AMPL (a mathematical programming language). To optimize the makespan, the total average waiting time and the used hosts on homogeneous cloud computing environments, Grandinetti et al. [28] solved their optimization problem based on the ϵ-constraint method. The approaches have been shown to be efficient. However, their implementation could be complex. The AMPL-based implementation requires to specify input data sets and variables to define the search space, and the ϵ-constraint method needs to choose suitable ϵ values. Compared to these, we will apply heuristic techniques to our optimization problem, which would make our approach simpler and easier to implement and deploy in a cloud computing system.

A large number of heuristics have been devised for cloud task scheduling in the past years. For instance, Su et al. [29] employed a cost-efficient task-scheduling algorithm by means of two heuristic strategies based on the idea of Pareto dominance. Besides that, some typical heuristic techniques such as clustering scheduling algorithm (e.g., DSC [30] and list scheduling algorithm (e.g., DSL [31]), have also been used in optimizating resource allocation in cloud. In contrast to these schemes, we focus on using metahuristics for cloud task scheduling, which is designed to find, generate, or select a heuristic that may provide a sufficiently good solution, especially with incomplete or imperfect information [32].

In fact, a trend of using metaheuristic algorithms is emerging rapidly in cloud computing [12], [33]. Various metaheuristic-based methods such as GA-based, ACO-based, PSO-based task scheduling algorithms have been proposed. Examples include but not limited to the following. Aziza et al. [34] proposed a time-shared and a space-shared genetic algorithm which are demonstrated to be able to outperform competed scheduling methods in terms of makespan and processing cost. Based on the ACO algorithm, Li et al. [35] introduced a load balancing algorithm for task scheduling in cloud computing. For PSO, Wang et al. [36] used an improved PSO algorithm to develop an optimal VM placement approach involving a tradeoff between energy consumption and global QoS guarantee for data-intensive services. To further improve the accuracy and efficiency of the above described metaheuristics in cloud computing, some works have tried to propose hybrid methods to leverage the strengths of the existing ones. Chen et al. [37] proposed a PSO-ACO method for task scheduling, showing it performs better than a standalone algorithm on makespan. To minimize task execution time, Liu et al. [38] presented a algorithm that makes use of the global search capability of genetic algorithm, and then converts the achieved results into the initial pheromone of ACO for further optimization. Moreover, Tsai et al. [39] proposed hyper-heuristic scheduling algorithm by integrating the GA, ACO and PSO, etc. into a single framework to reduce the makespan in cloud. Although all the approaches have demonstrated their advantages, different from them, we focus on exploring the application of the latest metaheuristics, the whale optimization algorithm [26], for cloud task scheduling. Moreover, we will try to use it on a multi-objective model to improve the performance of underlying computing systems.

Multi-objective optimization (MOO) is the process of simultaneously optimizing two or more conflicting objectives subject to a number of constraints [40]. In the context of cloud computing, the multi-objective optimization mainly includes the completion time, the constraints of QoS, energy consumption, economic cost, and the system performance [41]. Sheikhalishahi et al. [42] presented a scheduling system based on treating multi-resource optimization as multi-capacity bin packing. The solution is able to minimize the waiting time and the slowdown metrics. Ramezani et al. [43] tried to minimize task execution time, task transferring time, task execution cost and increase the QoS, using a multi-objective particle swarm optimization (MOPSO). Zuo et al. [41] introduced a model to optimize the makespan and resource cost on the basis of the ACO algorithm. In comparison, we will try to minimize the task execution time, system load and price cost using WOA.

To date, a lot of efforts have also been put on the designs of cloud scheduling systems. For example, Mao et al. [44] proposed an advanced scheduling strategy which could effectively shorten the time and maintain the stability of a system. Liu et al. [45] proposed a dependency-aware and resource-efficient scheduling which can achieve low response time and high resource utilization. In contrast to these, we focus on an algorithm design rather than system designs. On the other aspect, our approach can be applied to all the above designs to process tasks in cloud computing.

Generally, with the significant advantages on implementation, deployment as well as performance, metaheuristic algorithms have been widely studied on the optimization of cloud task scheduling in the past years. Although some research works have used the techniques on MOO in cloud computing, few of them focus on improving the performance of underlying computing systems. Moreover, none of them have ever applied the latest WOA on the MOO problem yet. In this work, we will try to minimize the task execution time, system load and price cost with a WOA-based method for cloud task scheduling. Moreover, to further improve the optimal solution search capability, we have also proposed several specified optimization for the proposed approach. To the best of our knowledge, this is the first work on applying WOA to multi-objective task scheduling problem in cloud computing.

III. Multi-objective Task Scheduling Model

In cloud computing, task scheduling policy will directly affect the efficiency of resource usage for underlying systems. Therefore, the allocation of input tasks to computing resources (e.g., VMs) becomes the key issue for cloud task scheduling. The logical view of a typical task scheduling process in a cloud computing system is illustrated in Fig. 1. There, the submitted jobs by users will be decomposed into a set of computing tasks first. We focus on the performance of different scheduling approaches in this paper, therefore we assume that all the tasks are logically independent of each other. Based on this, the process of task scheduling in cloud environment can be summarized as the following three steps. Firstly, based on the detailed information of the input tasks and the underlying available computing resources (e.g., VMs), tasks and resources will be mapped in accordance with a certain strategy. Then, following the mapping, the task scheduler at the schedule/control layer will generate an optimized task execution plan to meet the assigned requirements (i.e., the optimization objectives). Finally, the optimized plan is delivered to the underlying task processing layer (e.g., a cloud computing system) for execution, and the output results will be sent to the users.

TABLE I
Main Notations in Task Scheduling Model

Notation	Meaning
N	number of tasks to be processed
M	number of VMs
a_{ij}	decision variable to indicate whether the i-th task is assigned to the j-th VM
E_n (E_t)	processing capability vector for VMs (tasks)
S_n (S_t)	load capability vector for VMs (tasks)
C_n (C_t)	resource bandwidth vector for VMs (tasks)
P	price unit
w_i	weight of each cost function, $i \in \{1, 2, 3\}$

A. Task and Computing Resource Models

To describe the detailed optimization process of the scheduler, we use the following model under a cloud computing setting. There are a set of M computing nodes (VMs) $\{N_1, N_2, ..., N_m\}$ and a set of N computing task $\{T_1, T_2, ..., T_n\}$ with $N > M$, and the final scheduling result can be represented by a matrix A as following:

$$A_{nm} = \begin{bmatrix} a_{11} \, a_{12} \, \cdots a_{1m} \\ a_{21} \, a_{22} \, \cdots a_{2m} \\ \cdots \;\; \cdots \;\; \cdots \;\; \cdots \\ a_{n1} \, a_{n2} \, \cdots \, a_{nm} \end{bmatrix}$$

where a_{ij} is a decision variable that $a_{ij} = 1$ means that the i-th task is performed on the j-th VM , otherwise $a_{ij} = 0$, and there is $\sum_{j=1}^{M} a_{ij} = 1$ for each $i \in [1, N]$.

To characterize the general processing capability and resource consumption of a cloud computing system in a task scheduling scenario, we represent each resource node using three attributes. The first two are processing capability and load capability, which can be indicated by the CPU computing power and the memory size of a node respectively [41]. We employ the concept of *resource bandwidth* as the third attribute, to abstract the general recourse that a node can provide. The resource bandwidth of a node can be described by a function of its first two attributes, i.e., the larger the CPU power and memory size are, the larger the bandwidth will be.

In terms of the values of the three attributes, memory resources can be represented using megabytes. For the quantification of CPU resources, we specify the amount of CPU resources with a point-based system [46], such as that setting the full capacity of a single core with 100 points. Similarly, each computing task can be characterized by three attributes as well, i.e., the required CPU power, memory and resource bandwidth. On all these basis, we can model the underlying computing system as three vectors, i.e., the processing capability vector E_n, the load capability vector S_n and the resource bandwidth vector C_n. Similarly, three vectors are used for the tasks, i.e., E_t, S_t and C_t. For our presentation in the following, we use the notations as listed in Table I.

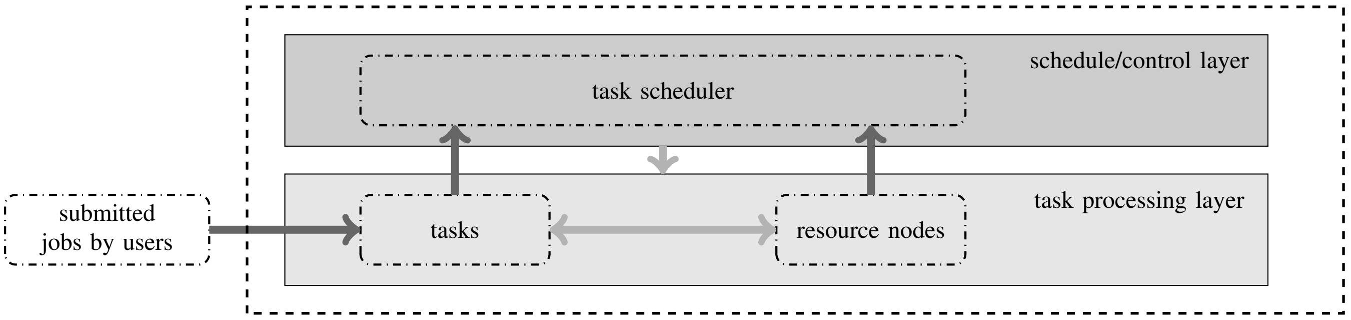


Fig. 1. A logical view of the task scheduling process in a cloud computing system.

B. Objective Functions

For a given set of tasks, it is expected that the underlying computing system can process the tasks in a highly efficient way, in terms of performance and resource consumption. Namely, the CPU power and memory of the system can be effectively used while the whole resource utilization cost can be minimized. Similar to the models with constrains on CPU and memory [47], the time cost function f_1 and the load cost function f_2 in our objectives are represented by Eq. (1) and Eq. (2), respetively. Moreover, the resource cost can be represented by some metrics such as energy consumption and economical cost [41]. Since they can be computed from resource bandwidth (such as with a very complex function), we just choose the price cost and use a price unit P in this work. Then, the price cost function f_3 can be represented by Eq. (3), where $E_{t,i}$ means the E_t value of the i-th task and $E_{n,j}$ is the E_n value of j-th VM. This representation is also similarly applied to the symbols S and C.

$$f_1 = \sum_{i=1}^{N}\sum_{j=1}^{M} a_{ij}\frac{E_{t,i}}{E_{n,j}} \tag{1}$$

$$f_2 = \sum_{i=1}^{N}\sum_{j=1}^{M} a_{ij}\frac{S_{t,i}}{S_{n,j}} \tag{2}$$

$$f_3 = \sum_{i=1}^{N}\sum_{j=1}^{M} a_{ij}\frac{E_{t,i}}{E_{n,j}} \times \frac{C_{t,i}}{C_{n,j}} \times P \tag{3}$$

In f_1, the time cost is calculated by summarizing the execution time of each task, which depends on the CPU power. We use the whole execution time rather than the makespan here, because we are more interested in task processing capability from a system angle rather than a service angle, and we assume that our system is highly efficient that a VM will be put into sleep when its assigned tasks have been done. The f_2 is computed on the basis of the required memory over the provided memory on each VM, the value of which is commonly used in simulation software to represent the load capability of a system, and a great value indicates a bad system load performance [47]. For a computing system, the price cost will not only depend on the task execution time, but also the ratio of the resource utilization at each time point. Therefore, in f_3, we add such a factor $C_{t,i}/C_{n,j}$ for each task on each VM when we calculate the whole cost. Namely, the price cost per time unit of a lightweight task (with a small value on resource bandwidth) will be less than a heavyweight task.

Obviously, our target to minimize the values of the above three functions is a MOO problem. The reason is that each of the functions has a different objective that can be conflicting. For example, we can speed up the processing of a task by using a powerful CPU, but the price cost would be increased. Also, for a case that a VM with a huge memory will be able to load a large number of tasks, but the whole task execution time could be long if its CPU computing power is low.

C. Optimization Model

To solve our MOO problem, we first normalize the matrices using the min–max normalization approach, and then represent the above three objective functions as F_1, F_2 and F_3 respectively, which are shown in below. The reason for this normalization is the values in E_n, S_n and C_n (also E_t, S_t and C_t) are in different scales, and the searching path for an optimal solution in this condition will be skewed, i.e., large values in a f_i will dominate the optimization process and the small ones would be totally ignored.

$$F_1 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} a_{ij}\frac{E_{t,i}/E_{n,j}}{\max\limits_{\forall i,j}\{E_{t,i}/E_{n,j}\}} \tag{4}$$

$$F_2 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} a_{ij}\frac{S_{t,ij}/S_{n,j}}{\max\limits_{\forall i,j}\{S_{t,i}/S_{n,j}\}} \tag{5}$$

$$F_3 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} a_{ij}\frac{(PE_{t,i}C_{t,i})\,/\,(E_{n,j}C_{n,j})}{\max\limits_{\forall i,j}\{(PE_{t,i}C_{t,i})\,/\,(E_{n,j}C_{n,j})\}} \tag{6}$$

Different cloud computing systems (or computing resource providers) could have different requirements on the performance of task executions. Therefore, similar to some recent works [34], [41], we employ some weight values (i.e., w_i) for the above three functions to make our target function tunable, which leads the final optimization objective function as:

$$F_{opt} = min \quad \{w_1F_1 + w_2F_2 + w_3F_3\} \tag{7}$$

The value of the weight w_i ($i \in \{1,2,3\}$) in Eq. 7 can be adjusted based on the requirements in practice. For example, in the scenarios such as the ones with a lightweight workload, we could be more interested in reducing the price cost of a computing system rather than the time and load cost. Then, we can set $w_1 = 0.25$, $w_2 = 0.25$ and $w_3 = 0.5$. In this condition, from a scheduling point of view, it is highly possible that a large number of input tasks will be allocated on economical VMs rather than the VMs with powerful CPU and large memory, since the improvement of time cost could be very limited for the latter case.

From the perspective of an optimization algorithm, to minimize the value of $(w_1F_1 + w_2F_2 + w_3F_3)$, the larger the value w_i is, the higher the priority of the algorithm on reducing the value of F_i will be. Specifically, when w_1 is much larger than w_2 and w_3, to reduce the value of F_1, it is more likely that all the tasks will be assigned to the VMs with more powerful CPUs. Similarly, if w_2 or w_3 is obviously larger, an optimization algorithm would assign the input tasks to the VMs with larger memory or resource bandwidth respectively. This kind of configuration could speed up the convergence of the searching process of an optimization algorithm, especially at its beginning phase, since the algorithm has the knowledge on priority for task assignment already. For an extreme case such as the setting with $w_1 = 1$, $w_2 = 0$ and $w_3 = 0$, the searching process on an optimal solution will be much simpler than other settings, since the scheduling problem is simplified to a single objective optimization problem. In this paper, for a general case, we just simply set $w_1 = w_2 = w_3 = \frac{1}{3}$. With this configuration, our optimization on the task scheduling problem in a cloud computing system can be represented as the Eq. (8) below:

$$F_{opt} = min \quad \{\frac{1}{3}(F_1 + F_2 + F_3)\} \tag{8}$$

IV. The Proposed Approach - IWC

In this section, we introduce how to apply the WOA algorithm to solve the optimization problem. Then, we propose the IWC with two optimization strategies to strengthen the searching capability of the WOA-based method.

A. The Whale Optimization Algorithm

In the WOA algorithm, a humpback whale in the search space is a candidate solution in the optimization problem, also called search agent, and the WOA utilizes a set of search agents to determine the possible or approximately global optimal solution. The searching process for a given problem begins with a set of random solutions, and the candidate solution is updated by the optimization rules until the end condition is met. The WOA algorithm can be divided into three main stages: encircling preying, bubble-net attack and search for prey. There mathematical representations are given as below.

1) Encircling Preying: In the initial stage, humpback whales do not know the optimal location in the search space when the prey is surrounded. In WOA, the current best solution is considered as the target prey and the whale closest to the prey is considered as the best search agent. Then, other individual whales may approach the target prey and gradually update their locations. This behavior is represented in the two functions below.

$$\vec{D} = |C \times \vec{X}^*(t) - \vec{X}(t)| \tag{9}$$

and

$$\vec{X}(t+1) = \vec{X}^*(t) - A \times \vec{D} \tag{10}$$

Here, $\vec{D}$ indicates the distance vector from the search agent to the target prey, t is the current iteration number, $\vec{X}^*$ is the local optimal solution and $\vec{X}$ is the position vector. $\vec{C}$ and $\vec{A}$ are the coefficient vectors and their calculations are defined as:

$$C = 2 \times r \tag{11}$$

and

$$A = 2a \times r - a \tag{12}$$

where r is a random number between 0 and 1, and a represents a linear decremented value from 2 to 0 based on the number of iteration t over the number of maximum iterations t_{max}, as shown below:

$$a = 2 - \frac{2t}{t_{max}} \tag{13}$$

2) Bubble-net Attack (exploitation phase): The behavior of whales' bubble-net attack is modeled based on the ideas of shrinking encircling and spiral position updating. We just briefly introduce their principles as below.

Shrinking encircling. From Eq. (10), we can see that the whales will shrink their encircling when $|A| < 1$. This means that the individual whales will approach the whale in the current best position, i.e., swim around the prey in a gradual contraction of a circle. The larger the value of $|A|$ is, the bigger steps the whales will take, and vice versa.

Spiral position updating. Each individual humpback whale first calculates its distance from the current optimal whale and then moves in a spiral shaped path. The mathematical model of the position update process is described as:

$$\vec{X}(t+1) = \vec{D}' \times e^{lb} \times \cos(2\pi l) + \vec{X}^*(t) \tag{14}$$

where $\vec{D}' = |\vec{X}^*(t) - \vec{X}(t)|$ is a vector indicating the distance from the individual whale to the best whale (current best found), b is a constant and l is a random number with the value between -1 and 1.

In order to mimic the two behaviors in a simultaneous way, it is assumed that the possibility of a whale updating its location based on the contraction path and the spiral path is 0.5 respectively, which can be described as

$$\vec{X}(t+1) = \begin{cases} \vec{X}^*(t) - A \times \vec{D} & p < 0.5 \\ \vec{D}' \times e^{lb} \times \cos(2\pi l) + \vec{X}^*(t) & p \geq 0.5 \end{cases} \tag{15}$$

where p is a randomly generated number between 0 and 1.

3) Search for Prey (exploration phase): To ensure that an approximately global optimal solution can be achieved, the search agents are pushed away from each other when $|A| > 1$. In this case, the position of the current optimal search agent will be replaced by a randomly selected search agent, and the responsible mathematical model is expressed as

TABLE II
TERMINOLOGY MAPPING BETWEEN WHALE PREDATION AND TASK SCHEDULING

Whale Predation	Task Scheduling
individual whale	cloud tasks
foraging process	optimal solution search process
whale position	a solution A_{nm} for F_{opt}
leader whale	optimal solution A_{nm} for F_{opt}
fitness of whale	value of F_{opt}

$$\vec{X}(t+1) = \vec{X}_{rand} - A \times |C \times \vec{X}_{rand} - \vec{X}(t)| \tag{16}$$

where $\vec{X}_{rand}$ is a position vector of the randomly selected search agent.

B. WOA-Based Task Scheduling

Our task scheduling problem described in Section III can be translated to the whale foraging problem with preemption as summarized in Table II: An individual whale corresponding to the given cloud tasks and the whale foraging process is the optimal solution searching process. In a search, a whale has a position corresponding to the scheduling problem has a solution A_{nm}, the position of the leader whale means the current optimal solution, and the fitness value of the leader whale is the current optimal value of the objective function F_{opt}. In such scenarios, we can use the WOA algorithm to get an optimized solution for our cloud task scheduling problem. Namely, in an iteration t of WOA, when all the whales update their positions, we can transfer the position information of each whale to a solution A_{nm} for given tasks. From the values in a matrix A_{nm}, we can compute the fitness value F_{opt} of the whale. The one with the smallest value will be considered for the leader whale, and its position information will be used for updating the positions of other whales in the next iteration (when required). All these processes will be repeated until the final iteration is reached. The position information of the final leader whale will be transferred to a matrix A_{nm}, and this solution will be used to generate the optimized task execution plan in the could computing system.

C. Optimization of Search Capability

Compared to many other advanced approaches, the WOA algorithm has several advantages. It is easy to implement and it only has a few parameters. To further improve the convergence speed and accuracy of our task scheduling process, we propose two optimization strategies.

1) Nonlinear Convergence Factor: In WOA, a random search agent is chosen for updating the positions of other agents when $|A| > 1$, and the best solution is selected for the case $|A| < 1$. To balance these exploration and exploitation characteristics, the factor a is employed to make ensure that the position of a newly updated search agent is in the range of $[-a, a]$. Based on Eq. (13), the value of a decreases linearly with the increase of the number of iterations. In this case, a large a in the early iterations is able to facilitate a global search and speed up the convergence of a search. In the meantime, a small one in the late iterations makes sure a local optimum can be achieved. However, this linear decrease could have two possible issues: (1) the accuracy of a search will be greatly impacted by the exploration if the global optimal value appears in an early iteration (as the $|A|$ will be still greater than 1); and (2) a local convergence could be slow once an optimal value is approached (because the step size would be very small for the non-optimal agents).

To remedy the above problems, we adopted a nonlinear convergence factor for our WOA-based task scheduling with updating the Eq. (13) to Eq. (17) as below.

$$a = (1 - \frac{t}{\gamma t_{max}})(1 + \frac{1}{1 - \gamma \frac{t}{t_{max}}}) \tag{17}$$

Here, γ is a parameter greater than 0. According to the above design, the value of a will increase sharply and then decrease quickly to a small value in the early iterations. After that, it will increase very slightly to the value of $1 - \frac{2}{\gamma}$. This makes sure that the search agents can conduct a very effective global searching at the beginning and then reach the possible optimal value in a quick way.

2) Adaptive Population Size: In a metaheuristic algorithm, a large population (with many search agents) can improve the accuracy on getting an optimal solution. However, it would impact the search performance when the searching space is small. Similarly, a small poplution would lead a suboptimal solution if the searching space is large. To improve this problem, some self-adaptive strategies on population size, in which the population pool size is either grown or shrunk every iteration based on the performance status of an algorithm, have been adapted in metahuristics [48].

In a WOA-based implementation, the number of whales is fixed in all the iterations. To further improve the performance of our task scheduling, we design a deterministic increase and delete operator based on the trigger rules as described in [49], and formulate our rules as following:

- *rule 1*: If the leader whale is continuously updated in 2 generations, and $ps > PS_{min}$, then the delete operator is executed to delete n_{dec} individual whales.
- *rule 2*: If the leader whale is not continuously updated in 1 generation, and $ps = PS_{max}$, then the delete operator is executed to delete n_{dec} individuals.
- *rule 3*: If the leader whale is not continuously updated in 1 generation, and $ps < PS_{max}$, then the increase operator is executed to add n_{inc} individuals.

Here, ps is the current population size, $PS_{\min}$ and PS_{max} are the lower and upper bound of the population size respectively. The number of whales to be increased or decreased is based on the Logistic model [50], which is used to describe population dynamics and general biological growth. The detailed designs of the two operators are as below.

Increase operators: When new individual whales are added, we should make sure that new information from the leader whale can be shared. In detail, the implementation can be mainly divided into three steps. The first is to determine

Algorithm 1 Whale Clustering

```
1: Generate a reference point $R$ within the search range
2: Among the current whales $P$, select a whale $X^{'}$ with the position closest to $R$
3: In $P\backslash\{X^{'}\}$, find out the points closest to $M-1$ and $X^{'}$ to form a sub-cluster
4: Delete $M$ individual whales in $P$
5: Repeat 2-4 until the whales have been divided into $N_p \backslash M$ classes.
```

the number of individuals to be increased based on Eq. (18) following [50]. The second step is to divide the population into n_{inc} groups using a general clustering approach as presented in Algorithm 1, and the optimal individual in each group is selected to form a set S. based on randomly selecting two individuals x_1 and x_2 from S, new individuals are generated in a cross way [51], as shown in Eq. (19), where $\alpha \in (0, 1)$.

$$n_{inc} = ps \times (PS_{\max} - ps)^2 \times PS_{\max}^{-2} \tag{18}$$

$$x_{new} = \alpha^{0.5} x_1 + (1 - \alpha^{0.5}) x_2 \tag{19}$$

In the above approach, new individual whales around the optimal value are added. This can improve the accuracy of the current optimal solution and also enhance the global development capability of the algorithm.

Decrease operators: Some redundant individuals could appear with the growth of the iterations. Similar to the increase operator, the decrease operator first determines the number of individuals to be deleted using Eq. (20). After that, the population is also divided into n_{dec} classes with Algorithm 1, and the worst individuals in each class are then deleted.

$$n_{dec} = ps^2 \times (PS_{\max} - ps) \times PS_{\max}^{-2} \tag{20}$$

In this case, the deleted individuals would be evenly distributed in the population, and thus the diversity of the population can be maintained.

D. Cloud Task Scheduling using IWC

Based on all the above designs, we summarize our IWC approach for cloud task scheduling as the flowchart demonstrated in Fig. 2. Its main implementation can be divided into the following four main steps:

- *Step 1*: This step focuses on the initialization of the implementation, which mainly includes the mapping between cloud computing tasks and the humpback whales, and the initialization of the positions of each individual whales. Moreover, some implementation parameters such as the search space dimension, the upper and lower bound of the whale population, and the maximum number of iterations are also initialized.
- *Step 2*: The optimal solution searching process based on WOA starts once the initialization is done. In this step, based on the position information, the fitness value of each whale is computed first. The whale with the smallest value, which is the current optimal solution, will be recorded. The global exploration and local exploitation behaviors of whales will depend on the value of the current A and also the random generated number p. If $p < 0.5$, whales will update their positions with a move around the current leader whale following the Eq. (14). Otherwise, the value of $|A|$ will be checked. For the case $|A| \geq 1$, the whales will update their positions with a randomly selected whale with Eq. (10), and they will swim around the leader whale in a circular way and update their positions using Eq. (10) when $|A| < 1$.
- *Step 3*: We apply our population control strategies to all the whales when their positions have been updated. The increase and decrease operators will follow the detailed rules as we have defined in Section IV-C2, i.e., whether the leader whale has continuously updated its position. The added or deleted whales will be based on the clustered whales and their numbers will be computed on the basis of the Eq. (18) and Eq. (20) respectively.
- *Step 4*: One iteration will be done when the positions of all the whales have been updated. The search process will terminate when the maximum number of iterations is reached, otherwise, it will go to *Step 2* for a new search. Once the assigned number of iterations has been reached, the position of the leader whale will be transferred to the decision variables a_{ij} as the best scheduling solution for the cloud computing tasks.

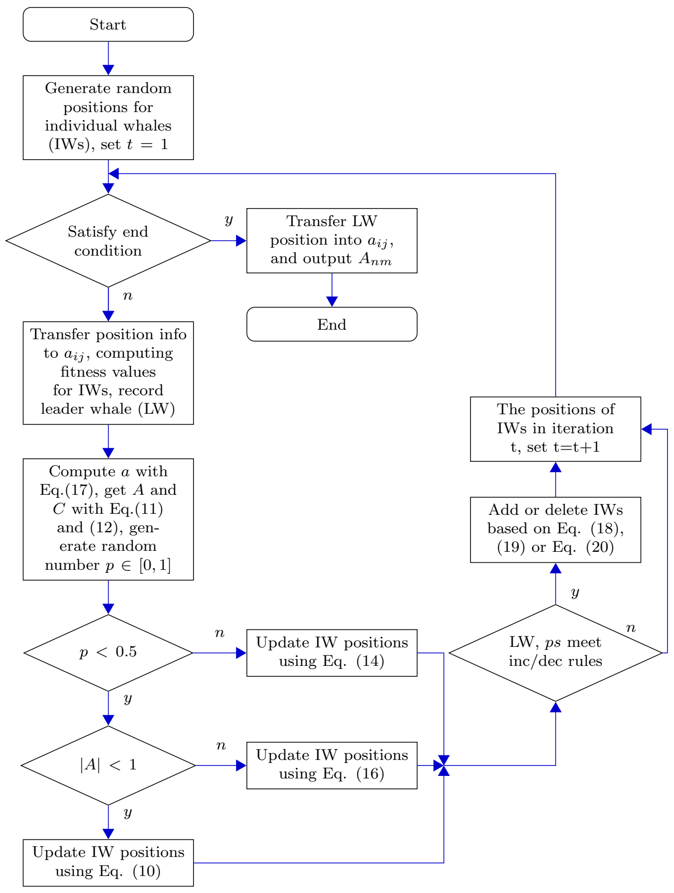

Start
Generate random positions for individual whales (IWs), set $t = 1$
Satisfy end condition
y
Transfer LW position into a_{ij}, and output A_{nm}
End
n
Transfer position info to a_{ij}, computing fitness values for IWs, record leader whale (LW)
Compute a with Eq.(17), get A and C with Eq.(11) and (12), generate random number $p \in [0,1]$
$p < 0.5$
n
Update IW positions using Eq. (14)
y
$|A| < 1$
n
Update IW positions using Eq. (16)
y
Update IW positions using Eq. (10)
LW, ps meet inc/dec rules
y
n
Add or delete IWs based on Eq. (18), (19) or Eq. (20)
The positions of IWs in iteration t, set t=t+1

Fig. 2. The implementation flow of IWC for cloud task scheduling.

TABLE III
MAIN PARAMETERS USED IN EXPERIMENTS

Algorithm	Parameter	Value	Description
ACO	ρ	0.7	pheromone evaporation coefficient
	p	0.3	path selection probability
PSO	w	0.9	inertia weight
	$c1$	1.8	acceleration constant
	$c2$	1.8	acceleration constant
IWC (part for WOA)	b	1	spiral searching path parameter
	$PS_{\max}$	51	the largest population
	$PS_{\min}$	10	initial population
	α	0.25	individual generate parameter
	γ	20	nonlinear factor

TABLE IV
PARAMETER SETUP OF VMS AND TASKS

Parameter	Value Range (VM)	Value Range (Task)
CPU E	$[200, 500]$	$[10, 50]$
Memory S	$[100, 500]$	$[50, 100]$
Resource C	$[100, 250]$	$[20, 50]$

To date, almost all the current metaheuristic algorithms have been applied to task scheduling in cloud computing. However, as one of the latest approaches, WOA has not been widely studied. Based on the WOA algorithm, our above implementation has provided an efficient solution for the task scheduling problem as described in Section III-C. Moreover, to enhance the search capability of the WOA-based scheduling, we have adopted two effective strategies to optimize the WOA parameters and control the population size during searching. As we will show in the experimental evaluation, our approach is indeed very effective on cloud task scheduling and can perform better compared to the current metaheuristic algorithms.

V. EXPERIMENTAL EVALUATION

In this section, we conduct a performance evaluation of the proposed IWC and compare it with some current approaches based on a set of simulation-based experiments.

A. Experimental Setup

We compare the IWC algorithm with some commonly used metaheuristic algorithms, i.e., ACO, PSO. Moreover, we also compare it with the described WOA-based scheduling in this work. We evaluate their performance on task scheduling in cloud computing based on simulations. We have implemented all the four algorithms using MatlabR2018b and all the code used in this work is available at https://github.com/longcheng11/IWC.

The evaluation metrics considered for performance analysis contain the cost for time, load and price as well as the total cost as we have described in our cost model in Section III. The evaluation is composed of two group experiments based on the number of scheduled tasks in the scenarios of small and large-scale computing. The main implementation parameters we have used in our tests for each algorithm are shown in Table III. The parameter configurations for VMs and tasks are shown in Table IV, and each parameter of a VM (task) is set to a random value from the ranges there. Additionally, we set the number of search agents (ants, particles and whales) to 50, the number of VMs to 40. Since a large number of iterations would bring in obvious significant for cloud task execution, similar to some recent works [4], [41], we have set the number of iterations to 100 in all our experiments.

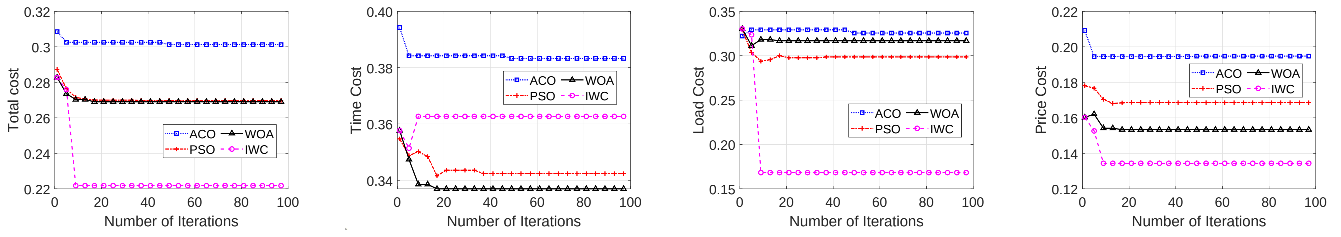
B. Experimental Results

To characterize the detailed performance of each algorithm, for each experiment, we report the achieved normalized values by increasing the number of iterations and the actual costs by varying the number of tasks.

1) Comparison with Small-scale Tasks: For the small-scale case, we increase the number of tasks from 100 to 1000. As a typical case, Fig. 3 shows the comparison of the four algorithms by increasing the number of iterations under the task 100. In Fig. 3(a), it can be seen that the (normalized) total cost of each algorithm decreases with the increase in the number of iterations, demonstrating the effectiveness of all the approaches on task scheduling in cloud computing. For the achieved best performance, the total cost of WOA and PSO is smaller than the ACO algorithm. This means that the two algorithms have better search capability on accuracy compared to ACO. In the meantime, we can see that IWC performs the best. Specifically, compared to the ACO, PSO and WOA, it reduces the their total cost by more than 20%. Moreover, with increasing the number of iterations, IWC can get its optimal solution in a quicker way than WOA, which means that our proposed optimization techniques can greatly improve the convergence speed of the WOA algorithm.

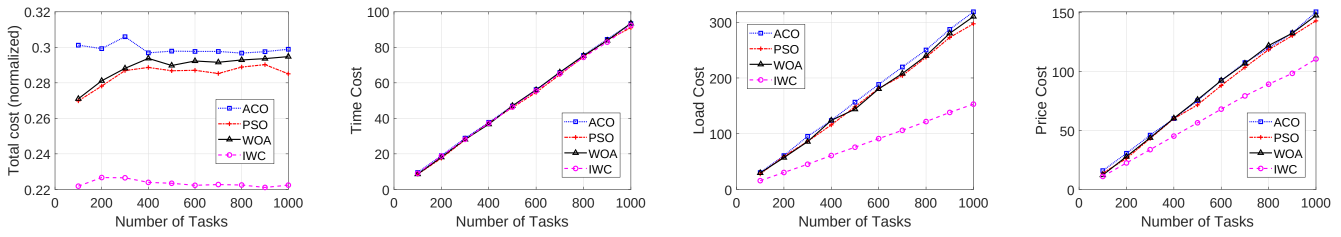
The results of the normalized load and price cost are shown in Fig. 3(c) and (d) respectively. The curves of the four algorithms there generally decrease with the increase of the number of iterations, showing that their cost can be effectively reduced during the optimal solution searching process. Similar to the total cost, IWC performs much better than the other three algorithms, in terms of convergence speed and accuracy. Compared to the three results in Fig. 3(a), (c) and (d), the time cost as presented in Fig. 3(b) are kind of different. It can be seen that the cost values of PSO, WOA and IWC fluctuate with the increase of iterations. From a global view, the ACO, PSO and WOA generally decrease while the IWC first decreases and then increases obviously to reach a stable status. This means that the first three approaches can use the CPU power in a more efficient way with the growing of the iterations. In comparison, IWC can not use the available CPU power in an optimal way. The reason for this could be that there is a trade-off between the local optimal value (i.e., time cost) and global optimal value (i.e., total cost) in the searching process. The IWC can reach a global optimal value based on a sacrifice of a local one, demonstrating again its strong capability on solving complex optimization problems.

The performance comparison of each algorithm with increasing the number of tasks is presented in Fig. 4. The normalized total cost in Fig. 4(a) shows that PSO and WOA perform better than ACO. However, IWC performs the best for all the tasks, indicating that it can effectively reduce the system cost for small-scale tasks in cloud computing. Fig. 4(b)-(d) present the actual cost of each algorithm on their



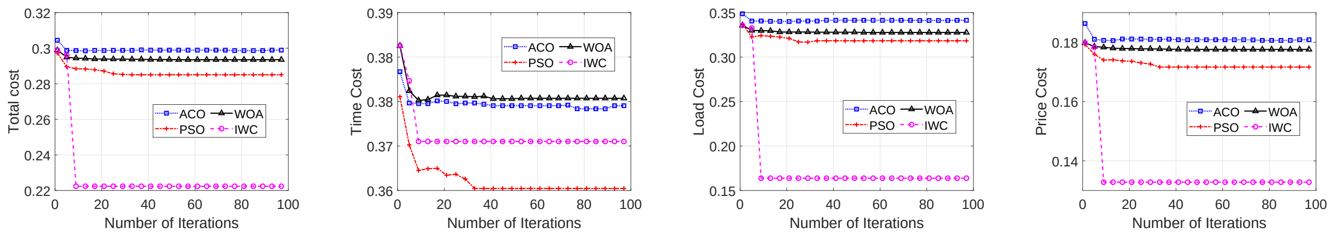
(a) the value of F_{obj} (b) the value of F_1 (c) the value of F_2 (d) the value of F_3

Fig. 3. The best performance (normalized values) achieved by increasing the number of iterations, for 100 tasks.



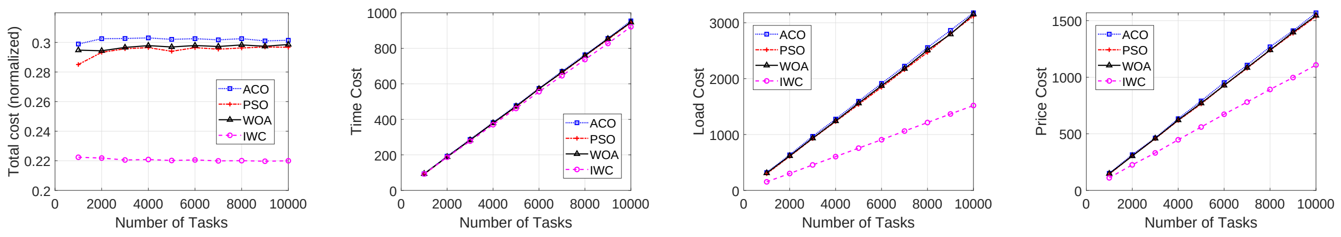
(a) the value of F_{obj} (b) the value of f_1 (c) the value of f_2 (d) the value of f_3

Fig. 4. The best performance achieved of each approach in the small-scale test.



(a) the value of F_{obj} (b) the value of F_1 (c) the value of F_2 (d) the value of F_3

Fig. 5. The best performance (normalized values) achieved by increasing the number of iterations, for 1000 tasks.



(a) the value of F_{obj} (b) the value of f_1 (c) the value of f_2 (d) the value of f_3

Fig. 6. The best performance achieved of each approach in the large-scale test.

task execution time, system load and price, respectively. We can see that all the values are increasing with increasing the number of tasks. This is reasonable, since the system workload increases. Regarding to the time cost, although the normalized values of the four algorithms have some differences, their actual values are nearly the same for each given task. This means that the computing power of the underlying system can be consumed in a similar way by the scheduled plans for all the four approaches. For the load and price cost, IWC can achieve obvious lower values than the other three algorithms. By increasing the number of tasks, the difference is more obvious. All these results show that IWC can greatly improve the utilization of system memory in a cloud computing environment. In the meantime, it can also obviously reduce the economical cost of resource utilization, compared to the current solutions.

2) *Comparison with Large-scale Tasks:* For the large-scale test, we gradually increase the number of tasks from 1000 to 10000. For the case with 1000 tasks, Fig. 5 shows the comparisons on the normalized values for all the four algorithms by increasing the number of iterations. Similar to the small-scale case, as illustrated in Fig. 5(a), the total cost of each algorithm decreases as the number of iterations increases, which demonstrates the advantages of all the approaches again

on cloud task scheduling. Moreover, it can be seen that IWC can achieve the best performance on the total cost, compared to the other three algorithms. This demonstrates again the effectiveness of the proposed IWC on handling large-scale task executions in cloud computing. With more details for the time cost shown in 5(b), the results are kind of different from the small-scale case in which IWC starts to perform better. This means that our approach can use system CPU power in a more efficient way in the presence of large task instances. Although IWC has a higher value than PSO here, it has a stronger capability on searching global optimal solution than PSO, when we consider the total cost. The results of the load and price cost of the four algorithms are shown in Fig 5(c) and (d), respectively. We can see that IWC can utilize the available system memory more efficiently, and it can also reduce more resource consumption for large-scale tasks. Specifically, compared to the other three methods, IWC can reduce the load cost by around 50% and about 30% on the price cost for underlying systems.

Fig. 6 presents the cost of the four algorithms by varying the number of tasks. For the normalized total cost as shown in Fig. 6(a), IWC always performs much better than other algorithms, which demonstrates the performance advantages of IWC on cloud task scheduling. Specifically, with the growth of the number of tasks, the normalized cost of IWC has slightly decrease while the other three approaches increase slightly, demonstrating that IWC has obvious advantages on searching optimal solutions in the presence of large or complex task workloads. For the actual cost on system time, load and price presented in Fig. 6(b)-(d), the curves of the four approaches are nearly the same as the results in the small-scale test. For example, all the values are increasing with increasing the number of tasks, and all the four algorithms perform generally the same for given tasks in terms of task execution time. The load and price cost of the proposed IWC algorithm is obviously smaller than the other three algorithms, and the differences become bigger with the increase in the number of tasks. All these demonstrate again that IWC can provide a obvious better solution on task scheduling in cloud computing, compared to the ACO, PSO and WOA algorithms.

VI. Conclusions

In this work, we have introduced a WOA-based task scheduling approach for cloud computing, which aims to improve the performance of a system with given computing resources. Moreover, to further improve the searching capability of the WOA-based scheduling, we have proposed the IWC approach with two advanced optimization strategies. We have presented the detailed implementation of IWC and our experimental results have shown that the proposed IWC is indeed efficient in terms of searching optimal scheduling plans. It can greatly improve the efficiency of a cloud computing system, in terms of both system load and system resource utilization cost, compared to some commonly used metaheuristic algorithms.

As the future work, to achieve better performance on convergence speed and accuracy in task scheduling, we will consider proposing more advanced strategies to further improve the balance between exploration and exploitation in the IWC approach. In the meantime, to reduce the scheduling overhead of the method in the presence of large workloads, we will explore the parallel implementations of our approach in cloud environments. Moreover, we will extend the proposed performance model and method with some more advanced features for task scheduling in cloud computing. For example, we will try to optimize the QoS problems, in which some tasks have higher priorities than others. Additionally, we also plan to use our approach to handle more complex task jobs, such as workflows [52], the tasks in which are not independent from each other, and cloud-based deap learning workloads [53]. Our long term goal is to develop a highly adaptive and efficient scheduling system for cloud computing in the presence of different task workloads.

References

[1] A. N. Toosi, R. N. Calheiros, and R. Buyya, “Interconnected cloud computing environments: Challenges, taxonomy, and survey,” *ACM Computing Surveys*, vol. 47, no. 1, p. 7, 2014.

[2] M. Cusumano, “Cloud computing and SaaS as new computing platforms,” *Communications of the ACM*, vol. 53, no. 4, pp. 27–29, 2010.

[3] S. K. Gavvala, C. Jatoth, G. Gangadharan, and R. Buyya, “QoS-aware cloud service composition using eagle strategy,” *Future Generation Computer Systems*, vol. 90, pp. 273–290, 2019.

[4] P. Kaur and S. Mehta, “Resource provisioning and work flow scheduling in clouds using augmented shuffled frog leaping algorithm,” *Journal of Parallel and Distributed Computing*, vol. 101, pp. 41–50, 2017.

[5] Z. Li, H. Shen, and C. Miles, “PageRankVM: A pagerank based algorithm with anti-collocation constraints for virtual machine placement in cloud datacenters,” in *Proc. 38th IEEE International Conference on Distributed Computing Systems*, 2018, pp. 634–644.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[7] H. Morshedlou and M. R. Meybodi, “Decreasing impact of sla violations: a proactive resource allocation approachfor cloud computing environments,” *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 156–167, 2014.

[8] B. Xu, C. Zhao, E. Hu, and B. Hu, “Job scheduling algorithm based on Berger model in cloud environment,” *Advances in Engineering Software*, vol. 42, no. 7, pp. 419–425, 2011.

[9] S. Kayalvili and M. Selvam, “Hybrid SFLA-GA algorithm for an optimal resource allocation in cloud,” *Cluster Computing*, pp. 1–9, 2018.

[10] L. Guo, S. Zhao, S. Shen, and C. Jiang, “Task scheduling optimization in cloud computing based on heuristic algorithm,” *Journal of networks*, vol. 7, no. 3, p. 547, 2012.

[11] 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.

[12] 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*, vol. 47, no. 4, p. 63, 2015.

[13] C. Blum and A. Roli, “Metaheuristics in combinatorial optimization: Overview and conceptual comparison,” *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[14] Z. Liang, X. Wang, Q. Lin, F. Chen, J. Chen, and Z. Ming, “A novel multi-objective co-evolutionary algorithm based on decomposition approach,” *Applied Soft Computing*, vol. 73, pp. 50–66, 2018.

[15] G. Kobeaga, M. Merino, and J. A. Lozano, “An efficient evolutionary algorithm for the orienteering problem,” *Computers & Operations Research*, vol. 90, pp. 42–59, 2018.

[16] M. A. Dulebenets, “A comprehensive evaluation of weak and strong mutation mechanisms in evolutionary algorithms for truck scheduling at cross-docking terminals,” *IEEE Access*, vol. 6, pp. 65 635–65 650, 2018.

[17] W. Du, M. Zhang, W. Ying, M. Perc, K. Tang, X. Cao, and D. Wu, “The networked evolutionary algorithm: A network science perspective,” *Applied Mathematics and Computation*, vol. 338, pp. 33–43, 2018.

[18] M. A. Dulebenets, “A delayed start parallel evolutionary algorithm for just-in-time truck scheduling at a cross-docking facility,” *International Journal of Production Economics*, vol. 212, pp. 236–258, 2019.

[19] B. Vahdani and S. Shahramfard, “A truck scheduling problem at a cross-docking facility with mixed service mode dock doors,” *Engineering Computations*, 2019.

[20] E. Gafarov and F. Werner, “Two-machine job-shop scheduling with equal processing times on each machine,” *Mathematics*, vol. 7, no. 3, p. 301, 2019.

[21] 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.

[22] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*. John Wiley & Sons, 2000, vol. 7.

[23] M. Dorigo and G. Di Caro, “Ant colony optimization: a new meta-heuristic,” in *Proc. 1999 Congress on Evolutionary Computation*, vol. 2, 1999, pp. 1470–1477.

[24] J. Kennedy, “Particle swarm optimization,” in *Encyclopedia of machine learning*. Springer, 2011, pp. 760–766.

[25] W.-z. Sun, J.-s. Wang, and X. Wei, “An improved whale optimization algorithm based on different searching paths and perceptual disturbance,” *Symmetry*, vol. 10, no. 6, p. 210, 2018.

[26] S. Mirjalili and A. Lewis, “The whale optimization algorithm,” *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016.

[27] M. Malawski, K. Figiela, and J. Nabrzyski, “Cost minimization for computational applications on hybrid cloud infrastructures,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1786–1794, 2013.

[28] L. Grandinetti, O. Pisacane, and M. Sheikhalishahi, “An approximate ϵ-constraint method for a multi-objective job scheduling in the cloud,” *Future Generation Computer Systems*, vol. 29, no. 8, pp. 1901–1908, 2013.

[29] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, “Cost-efficient task scheduling for executing large programs in the cloud,” *Parallel Computing*, vol. 39, no. 4-5, pp. 177–188, 2013.

[30] T. Yang and A. Gerasoulis, “DSC: Scheduling parallel tasks on an unbounded number of processors,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 9, pp. 951–967, 1994.

[31] G. C. Sih and E. A. Lee, “A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures,” *IEEE transactions on Parallel and Distributed systems*, vol. 4, no. 2, pp. 175–187, 1993.

[32] Metaheuristic, https://en.wikipedia.org/wiki/Metaheuristic.

[33] C.-W. Tsai and J. J. Rodrigues, “Metaheuristic scheduling for cloud: A survey,” *IEEE Systems Journal*, vol. 8, no. 1, pp. 279–291, 2014.

[34] H. Aziza and S. Krichen, “Bi-objective decision support system for task-scheduling based on genetic algorithm in cloud computing,” *Computing*, vol. 100, no. 2, pp. 65–91, 2018.

[35] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang, “Cloud task scheduling based on load balancing ant colony optimization,” in *2011 Sixth Annual ChinaGrid Conference*. IEEE, 2011, pp. 3–9.

[36] S. Wang, A. Zhou, C.-H. Hsu, X. Xiao, and F. Yang, “Provision of data-intensive services through energy-and qos-aware virtual machine placement in national cloud data centers.” *IEEE Trans. Emerging Topics Comput.*, vol. 4, no. 2, pp. 290–300, 2016.

[37] X. Chen and D. Long, “Task scheduling of cloud computing using integrated particle swarm algorithm and ant colony algorithm,” *Cluster Computing*, pp. 1–9, 2017.

[38] C.-Y. Liu, C.-M. Zou, and P. Wu, “A task scheduling algorithm based on genetic algorithm and ant colony optimization in cloud computing,” in *Proc. 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science*, 2014, pp. 68–72.

[39] C.-W. Tsai, W.-C. Huang, M.-H. Chiang, M.-C. Chiang, and C.-S. Yang, “A hyper-heuristic scheduling algorithm for cloud,” *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 236–250, 2014.

[40] E. Demir, T. Bektaş, and G. Laporte, “The bi-objective pollution-routing problem,” *European Journal of Operational Research*, vol. 232, no. 3, pp. 464–478, 2014.

[41] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, “A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing,” *IEEE Access*, vol. 3, pp. 2687–2699, 2015.

[42] M. Sheikhalishahi, R. M. Wallace, L. Grandinetti, J. L. Vazquez-Poletti, and F. Guerriero, “A multi-dimensional job scheduling,” *Future Generation Computer Systems*, vol. 54, pp. 123–131, 2016.

[43] F. Ramezani, J. Lu, and F. Hussain, “Task scheduling optimization in cloud computing applying multi-objective particle swarm optimization,” in *International Conference on Service-oriented computing*, 2013, pp. 237–251.

[44] Y. Mao, V. Green, J. Wang, H. Xiong, and Z. Guo, “DRESS: dynamic resource-reservation scheme for congested data-intensive computing platforms,” in *Proc. 11th IEEE International Conference on Cloud Computing*, 2018, pp. 694–701.

[45] J. Liu and H. Shen, “Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds,” in *Proc. 2016 IEEE CloudCom*, 2016, pp. 110–117.

[46] X. Liu and R. Buyya, “Resource management and scheduling in distributed stream processing systems: A taxonomy, review and future directions,” *ACM Computing Surveys*, vol. 1, no. 1, 2018.

[47] Y. Hu, C. De Laat, Z. Zhao *et al.*, “Multi-objective container deployment on heterogeneous clusters,” in *Proc. 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2019, pp. 592–599.

[48] S.-T. Hsieh, T.-Y. Sun, C.-C. Liu, and S.-J. Tsai, “Solving large scale global optimization using improved particle swarm optimizer,” in *Proc. 2008 IEEE Congress on Evolutionary Computation*, 2008, pp. 1777–1784.

[49] R.-F. Wang, L.-C. Jiao, F. Liu, and S.-Y. Yang, “Nature computation with self-adaptive dynamic control strategy of population size,” *Journal of Software*, vol. 23, no. 7, pp. 1760–1772, 2012.

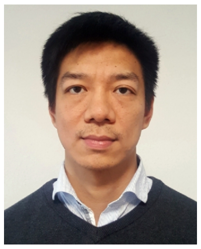
[50] A. Tsoularis and J. Wallace, “Analysis of logistic growth models,” *Mathematical biosciences*, vol. 179, no. 1, pp. 21–55, 2002.

[51] M. S. Arumugam and M. Rao, “On the improved performances of the particle swarm optimization algorithms with adaptive parameters, cross-over operators and root mean square (rms) variants for computing optimal control of a class of hybrid systems,” *Applied Soft Computing*, vol. 8, no. 1, pp. 324–336, 2008.

[52] L. Cheng, B. Van Dongen, and W. Van Der Aalst, “Scalable discovery of hybrid process models in a cloud computing environment,” *IEEE Transactions on Services Computing*, 2019.

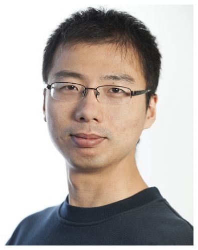
[53] W. Zheng, M. Tynes, H. Gorelick, Y. Mao, L. Cheng, and Y. Hou, “Flowcon: Elastic flow configuration for containerized deep learning applications,” in *Proc. 48th International Conference on Parallel Processing*, 2019, pp. 87:1–87:10.

Xuan Chen an Associate Professor with Design and Art, Zhejiang Industry Polytechnic College, Shaoxing, China. He received the B.S. degree in information management and information system from Zhengzhou University of Aeronautics, Zhengzhou, China in 2003 and received the M.S. degree in computer Software engineering from University of Electronic Science and Technology of China, Chengdu, China in 2011. His research interests include cloud computing and algorithm design.



Long Cheng is an Assistant Professor in the School of Computing at Dublin City University, Ireland. He received the B.E. from Harbin Institute of Technology, China in 2007, M.Sc from University of Duisburg-Essen, Germany in 2010 and Ph.D from National University of Ireland Maynooth in 2014. He was a Marie Curie Fellow at University College Dublin. He has worked at organizations such as Huawei Technologies, IBM Research, TU Dresden and TU Eindhoven. His research focuses on high performance data analytics, distributed systems and process mining. He is a member of the IEEE.

Cong Liu is a Professor with the Shandong University of Technology, China. He received the B.S. and M.S. degrees in computer software and theory from the Shandong University of Science and Technology, Qingdao, China, in 2013 and 2015, respectively, and the Ph.D. degree in computer science and information systems from the Eindhoven University of Technology, The Netherlands, in 2019. His current research interests include business process management, process mining, Petri nets and big data.



Qingzhi Liu is a Lecturer at the Information Technology Group, Wageningen University, The Netherlands. He received a B.E. and a M.Eng. from Xidian University, China in 2005 and 2008 respectively. He received a M.Sc. (with cum laude) and a Ph.D. from Delft University of Technology, The Netherlands in 2011 and 2016 respectively. He was a Postdoctoral Researcher at the System Architecture and Networking Group, Eindhoven University of Technology, The Netherlands from 2016 to 2019. His research interests include Internet of Things and machine learning.

Jinwei Liu is an Assistant Professor in the Department of Computer and Information Sciences at Florida A&M University. He received the M.S. degree in Computer Science from Clemson University and University of Science and Technology of China. He received his Ph.D. degree in Computer Engineering from Clemson University in 2016. He has worked at University of Virginia and University of Central Florida. His research interests include cloud computing, big data, machine learning and data mining, cybersecurity, wireless sensor networks, social networks, HPC and IoT. He is a member of the IEEE and the ACM.

Ying Mao is an Assistant Professor in the Department of Computer and Information Science at Fordham University in the New York City. He received his Ph.D. in Computer Science from the University of Massachusetts Boston in 2016. His research interests mainly focus on the fields of cloud computing, virtualization, resource management, and data-intensive platforms.

John Murphy is a Professor at University College Dublin (UCD). He is an IBM Faculty Fellow, a Fellow of the IET, a Senior Member of the IEEE, a Fellow and Chartered Engineer with Engineers Ireland, and a Fellow of the Irish Computer Society. For many years he held an academic part-time position at the Jet Propulsion Laboratory in Pasadena, and acted as a consultant to the US Department of Justice. He has published over 200 peer-reviewed journal articles or international conference full papers in performance engineering of networks and distributed systems. He has supervised 24 Ph.D. students to completion and been awarded over 30 competitive research grants.