Hierarchical Multi-Agent Optimization for Resource Allocation in Cloud Computing

Xiangqiang Gao, Rongke Liu, *Senior Member, IEEE*, and Aryan Kaushik, *Member, IEEE*

Abstract—In cloud computing, an important concern is to allocate the available resources of service nodes to the requested tasks on demand and to make the objective function optimum, i.e., maximizing resource utilization, payoffs, and available bandwidth. This article proposes a hierarchical multi-agent optimization (HMAO) algorithm in order to maximize the resource utilization and make the bandwidth cost minimum for cloud computing. The proposed HMAO algorithm is a combination of the genetic algorithm (GA) and the multi-agent optimization (MAO) algorithm. With maximizing the resource utilization, an improved GA is implemented to find a set of service nodes that are used to deploy the requested tasks. A decentralized-based MAO algorithm is presented to minimize the bandwidth cost. We study the effect of key parameters of the HMAO algorithm by the Taguchi method and evaluate the performance results. The results demonstrate that the HMAO algorithm is more effective than two baseline algorithms of genetic algorithm (GA) and fast elitist non-dominated sorting genetic algorithm (NSGA-II) in solving the large-scale optimization problem of resource allocation. Furthermore, we provide the performance comparison of the HMAO algorithm with two heuristic Greedy and Viterbi algorithms in on-line resource allocation.

Index Terms—Cloud computing, resource allocation, resource utilization, bandwidth cost, genetic algorithm, multi-agent optimization

1 Introduction

For some electronic devices, which are composed of dedicated hardware equipments, i.e., field programmable gate array (FPGA), digital signal processor (DSP) and integrated circuit (IC), the compatibilities for different requested tasks are difficult to guarantee and the systems will be more complicated with the increase in the number of the requested tasks. The software defined network (SDN) and virtualization technology are the foundations of the cloud computing, and provide a promising and flexible approach to facilitate resource allocation [1], [2], [3]. Cloud service providers can allocate the available resources related to service nodes to the requested tasks depending on demand and supply. When a task consists of multiple sub-tasks, these sub-tasks could be deployed on several service nodes and form a service chain, which is a data flow through the service nodes in sequence and can be presented as a directed acyclic graph (DAG) [4]. Each sub-task needs the physical resources for central processing unit (CPU), memory, or graphic processing unit (GPU). Besides, there are bandwidth costs to transfer data on different service nodes. For example, in case of data transmission, it includes five sub-tasks and the service chain about these sub-tasks can be represented as: network receiving → capture → tracking → synchronization → decoding, where each functional module is achieved by software programming and can run on a commonly used computer system. The complexity and development cost of a system can be effectively reduced by cloud computing, and the flexibility and scalability can also be improved. However, a new challenge in cloud computing is how to effectively allocate the available resources related to service nodes to the requested tasks, which leads to a combinatorial optimization problem [5], [6].

1.1 Literature Review

The optimization problems for resource allocation in cloud computing have been widely studied [7], [8], [9], [10], [11], which are proved to be NP-hard and the complexities are analyzed in [12], [13], [14]. Meta-heuristic algorithms are effective optimization approaches for solving these resource allocation problems. Several variants of genetic algorithm (GA) are developed to improve the performance of the resource allocation solution in [7], [8], [9] and the fast elitist non-dominated sorting genetic algorithm (NSGA-II), which is described in detail in [15], is also used to tackle this problem [10], [11]. Two modified particle swarm optimizations (MPSO) are proposed to reallocate the migrated virtual machines and achieve the resource management based on a flexible cost in [16] and [17], respectively. Moreover, an ant colony optimization (ACO) for dealing with the nonlinear resource allocation problem is presented in [18]. To enhance the efficiency in terms of seeking the optimal solution, the authors in [19] introduce a hybrid optimization algorithm of simulated annealing and artificial bee colony (ABC-SA).

However, as the scale of the optimization problem grows, a large feasible solution space needs to be searched and the computational complexity order of seeking the optimal solution increases. Hence, the performance of solving

- *Xiangqiang Gao and Rongke Liu are with the School of Electronic and Information Engineering, Beihang University, Beijing 100191, China. E-mail: {xggao, rongke_liu}@buaa.edu.cn.*
- *Aryan Kaushik is with the Department of Electronic and Electrical Engineering, University College London (UCL), WC1E 7JE London, U.K. E-mail: a.kaushik@ucl.ac.uk.*

Manuscript received 26 Nov. 2019; revised 2 Sept. 2020; accepted 9 Oct. 2020. Date of publication 14 Oct. 2020; date of current version 22 Oct. 2020. (Corresponding author: Rongke Liu.) Recommended for acceptance by O. Rana. Digital Object Identifier no. 10.1109/TPDS.2020.3030920

the problem could be reduced by using meta-heuristic algorithms [20], [21]. To further solve this issue, some optimization algorithms based on decomposition and cooperative co-evolutionary method are introduced, where a large-scale problem is divided into several small-scale problems and global optimal solution can be obtained by addressing these sub-problems with cooperative co-evolutionary method [22], [23]. Reference [24] proposes a new cooperative co-evolution framework (CCFR), which can efficiently allocate computational resources based on the contributions of different sub-populations, to address a large-scale optimization problem.

Compared with centralized optimization methods, distributed optimization algorithms based on multi-agent systems (MAS) [25] have an explicit potential advantage for solving the task deployment and resource allocation in cloud computing [26], [27], [28]. In [26], the product allocation problem for supply chain market is considered as a discrete resource allocation and solved by a multi-agent based distributed optimization algorithm. In addition, an efficient greedy algorithm with multi-agent is proposed to address the task allocation problem in social networks [27], where the agents just require their local information about tasks and resources, and provide the resources for the tasks by an auction mechanism. Another auction-based virtual machine resource allocation approach with the multi-agent system is presented to save energy cost, and the virtual machines assigned on different agents can be exchanged by a local negotiation-based approach in [28].

In our work, a hierarchical multi-agent optimization (HMAO) algorithm is proposed to address the problem of maximizing the resource utilization and making the bandwidth cost minimum. The proposed HMAO algorithm consists of improved GA and multi-agent optimization (MAO) algorithm. To reduce the complexity of solving the objective problem, we decompose it into two sub-objective problems: maximizing the resource utilization and minimizing the bandwidth cost. The improved GA is used to obtain the optimal set of service nodes that are used by the requested tasks. The MAO algorithm is proposed to minimize the bandwidth cost. For the MAO algorithm, we design a shared agent and several service agents. Shared agent can hold the information about resource allocation and task deployment for all service nodes. Service agents can aware the environment of cloud computing by accessing the shared agent. In addition, we also implement the selection and exchange operators by a probabilistic method to migrate and swap the sub-tasks on the service nodes. According to the proposed HMAO algorithm, we can improve the performance of solving resource allocation as the number of the requested tasks increases.

1.2 Contributions

In this paper, we assume that the information about the requested tasks and service nodes is given in advance, e.g., task types, the number of tasks, resource requirements for each sub-task and resource capacity values of service nodes. Furthermore, we formulate the problem of resource allocation to maximize the resource utilization and minimize the bandwidth cost under resource constraints. The resource utilization is defined as the ratio of the number of resources used by all the requested tasks to the total number of resources of the service nodes that are used to deploy the requested tasks. In order to address the optimization problem, we propose a hierarchical multi-agent optimization (HMAO) algorithm which is a combination of improved GA and multi-agent optimization (MAO) algorithm.

First, we decompose the main objective of maximizing the resource utilization and minimizing the bandwidth cost into two sub-objectives: maximizing the resource utilization and minimizing the bandwidth cost [24]. The two sub-objective optimization problems are in conflict with each other and we assume that the former is considered with a higher priority. The improved GA is used to seek the optimal solution in order to maximize the resource utilization and the optimal solution can be expressed as a set of service nodes that are used to deploy the requested tasks. For the MAO algorithm, there are two types of agents: service agent and shared agent. Service agents are assigned to each service node to assist in resource management. A shared agent holds the information about resource allocation for all the service nodes and supports the service agents in *access* and *update* processes. The agents have environment-aware, autonomy, social behavior and load-balancing properties [29]. The service agents visit the shared agent to obtain the information about resource allocation for all the service agents, and they can migrate and swap their sub-tasks with each other by selection and exchange operators that are designed based on a probabilistic approach. In addition, a priority-based source sub-task selection mechanism for the selection operator is implemented by considering load-balancing on the service nodes and the relationships between different sub-tasks. As a result, a global optimal solution will be obtained by seeking the optimal solutions of the service agents with cooperative co-evolutionary method [23]. The proposed HMAO algorithm provides the following contributions.

1) Considering to solve the joint problem of optimizing the resource utilization and bandwidth cost as an entire problem, it will increase the computational complexity of the optimization problem and weaken the performance of seeking the optimal solution, especially, for the high-dimensional problems. In this paper, we decompose the total optimization problem into two sub-problems. Correspondingly, a hierarchical multi-agent optimization algorithm, which combines an improved GA with MAO algorithm, is presented for solving the two optimization sub-problems. So that we can effectively reduce the computational complexity of the overall optimization problem.
2) The set of available service nodes, which represents the optimal solution of maximizing the resource utilization, can be obtained by the improved GA. Then the MAO algorithm is proposed to solve the sub-problem of minimizing the bandwidth cost. We consider four main characteristics for the agents: environment-aware, autonomy, social behavior and load-balancing. Furthermore, we design the action sets and behavior criteria for service agents and a shared agent, the relationships between different

agents are illustrated based on an organized architecture. To migrate and swap those sub-tasks on the service nodes, we implement two operators of selection and exchange, where the selection operator consists of a source sub-task selection and a target sub-task selection. Considering load-balancing on the service nodes and the relationships between different sub-tasks, the source sub-task is provided by a priority-based selection mechanism. For the MAO algorithm, a feasible solution is partitioned into several small-scale solutions and each service agent indicates one part. We can find the global optimal solution by optimizing the objectives of the service agents with cooperative co-evolutionary method.

3) To keep diversity in feasible solutions and avoid premature convergence, the selection and exchange operators for the MAO algorithm are implemented based on a probabilistic method. For the former, we can randomly choose a sub-task from the service nodes as the target sub-task through the selection probability. Similarly, if the objective result for a service agent does not improve after the exchange, it can also continue to be executed with the use of exchange probability. By introducing a probabilistic method to the selection and exchange operators, we can further improve the performance of the proposed HMAO algorithm.

Finally, the experiments for different tasks are carried out to verify the performance of the proposed HMAO algorithm. The results show that the proposed HMAO algorithm is an effective approach to solve the optimization problem of resource allocation. When compared with GA and NSGA-II, the proposed HMAO algorithm performs better for high-dimensional problems in terms of solution quality, convergence time and stability.

The remainder of this paper is organized as follows. Section 2 introduces the system model of resource allocation. In Section 3, we provide the problem formulation for the system model. A hierarchical multi-agent optimization algorithm is proposed in Section 4. Section 5 investigates the effect of key parameters for the proposed HMAO algorithm and evaluates the performance comparison with existing optimization algorithms. The conclusion of this paper is discussed in Section 6.

2 System Model

A cloud computing system can be modeled as a graph $G = < V, E >$, where $V = \{v_0, v_1, \ldots, v_{K-1}\}$ represents a set of service nodes with K service nodes, v_i is the ith service node. E describes a set of links between these service nodes and $< v_i, v_j > \in E$ indicates the link between service nodes v_i and v_j. Service node v_i contains four resources: CPU, memory, GPU and bandwidth, whose capacities are denoted as C^i, M^i, G^i and B^i, respectively. We also assume that any two service nodes can communicate with each other through inter-connected networks, i.e., $\forall < v_i, v_j > \in E$. Let $T = \{t_0, t_1, \ldots, t_{L-1}\}$ be a set of tasks with L total number of tasks and t_l denotes the lth task. t_l consists of N_l sub-tasks and is expressed as $t_l = \{t_{l,0}, t_{l,1}, \ldots, t_{l,N_l-1}\}$, which is an ordered list and there is a precedence relationship between different sub-tasks. That is, $t_{l,n}$ can not be carried out until all its predecessors are finished. The resource requirements of CPU, memory, GPU and bandwidth for $t_{l,n}$ running on v_i are described as $c^i_{l,n}, m^i_{l,n}$, $g^i_{l,n}, b^i_{l,n}$, respectively. In cloud computing, an effective resource allocation approach is used to allocate the available resources of service nodes to the requested tasks. Note that the total number of resources on any service node can not be more than its capacities, and the bandwidth cost will be considered when two adjacent sub-tasks are placed on different service nodes. Moreover, we can deploy as many sub-tasks as possible to a service node in order to improve the resource utilization [7], [11], [12].

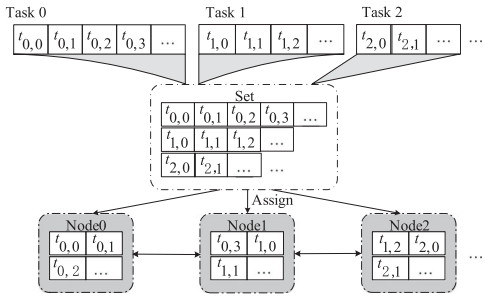


Fig. 1. Procedure for resource allocation.

The procedure for resource allocation in our system model is shown in Fig. 1, where the information about the requested tasks and service nodes is obtained in advance, all sub-tasks from the requested tasks are deployed to multiple service nodes simultaneously, and the neighbouring sub-tasks can be located on the same service node or different ones. Therefore, different resource allocation schemes have an impact on the system performance [12].

Fig. 2 presents an example of resource allocation with different schemes. There are five service nodes with communication links in the network, two tasks are given as $t_0 = \{t_{0,0}, t_{0,1}, t_{0,2}, t_{0,3}, t_{0,4}\}$ and $t_1 = \{t_{1,0}, t_{1,1}, t_{1,2}, t_{1,3}\}$. In Fig. 2a, for task t_0, sub-tasks $t_{0,0}, t_{0,1}$ are deployed on service node v_0, sub-tasks $t_{0,2}, t_{0,3}$ and $t_{0,4}$ are on service nodes v_1, v_3 and v_4, respectively. A service chain for t_0 is built on the service nodes v_0, v_1, v_3 and v_4 in order, and there is no bandwidth cost between sub-tasks $t_{0,0}$ and $t_{0,1}$ as they are on the same service node. Hence, the bandwidth cost for t_0 is $b^0_{0,0} + b^1_{0,2} + b^3_{0,3} + b^4_{0,4}$. For task t_1, sub-tasks $t_{1,0}$ and $t_{1,1}$ are allocated on service node v_2, and sub-tasks $t_{1,2}$ and $t_{1,3}$ are on service nodes v_3 and v_4, respectively. The bandwidth cost for t_1 is indicated as $b^2_{1,0} + b^3_{1,2} + b^4_{1,3}$. Different resource allocation schemes for tasks t_0 and t_1 are described in Fig. 2b. sub-tasks $t_{0,0}, t_{0,1}$ and $t_{0,2}$ are assigned on service node v_0, and sub-tasks $t_{0,3}$ and $t_{0,4}$ are on service nodes v_1 and v_4,

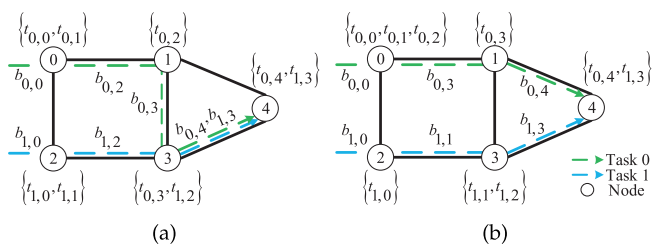


Fig. 2. Example of resource allocation with different schemes.

respectively, and the bandwidth cost for t_0 is $b^0_{0,0} + b^1_{0,3} + b^4_{0,4}$. Similarly, the bandwidth cost for t_1 is $b^2_{1,0} + b^3_{1,1} + b^4_{1,3}$. As shown in Fig. 2, the bandwidth costs vary for different resource allocation schemes.

3 Problem Formulation

In this section, a mathematical description of resource allocation is presented for our system model. Our purpose is to maximize the resource utilization and minimize the bandwidth cost in cloud computing with several physical constraints. Moreover, we do not consider the network resource constraints in this paper, such as routers and switches.

Note that several redundant service nodes are placed to maintain service delivery by using the proposed approach in a practical cloud environment. In general, these redundant service nodes are in sleep or shutdown states to reduce the operating cost [30], [31]. When a few service nodes fail to handle their dynamical workloads, the redundant service nodes will be woken up quickly. The dynamical workloads in the failed service nodes can be migrated to the redundant service nodes, so it ensures that our proposed approach can be also effective and maintain service delivery in the case of service node failure.

The main symbols used to formulate our problem are summarized in Table 1.

To further discuss the problem, we define a binary decision variable $x^i_{l,n}$ that indicates whether sub-task $t_{l,n}$ is deployed on service node v_i, i.e., $x^i_{l,n} = 1$ means $t_{l,n}$ is allocated on v_i, otherwise not. In addition, another binary decision variable $y^{i,j}_{l,n,\hat{n}}$ is used to describe whether there is a bandwidth cost between sub-tasks $t_{l,n}$ and $t_{l,\hat{n}}$. Let us denote a set of predecessors of $t_{l,n}$ as $U^p_{l,n}$ and a set of successors of $t_{l,\hat{n}}$ as $U^s_{l,\hat{n}}$. We assume that $t_{l,n}$ and $t_{l,\hat{n}}$ are placed on v_i and v_j, respectively. When $U^p_{l,n} \neq \varnothing, t_{l,\hat{n}} \in U^p_{l,n}$, then $y^{i,j}_{l,n,\hat{n}} = 1$, or else $y^{i,j}_{l,n,\hat{n}} = 0$. Note that the bandwidth cost between a sub-task and the source node is not neglected. The bandwidth cost for $t_{l,n}$ can be written as

$$\tilde{b}^i_{l,n} = \begin{cases} b^i_{l,n}, & \text{if } U^p_{l,n} = \varnothing \quad \text{or} \quad y^{i,j}_{l,n,\hat{n}} = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

With the physical resource constraints [9], the number of resources used by the requested tasks on a service node should be less than the resource capacities. As a result, the resource constraints about CPU, memory and GPU are given as follows:

$$\begin{cases} \sum\limits_{l=0}^{L-1} \sum\limits_{n=0}^{N_l-1} c^i_{l,n} x^i_{l,n} \leq C^i, \\ \sum\limits_{l=0}^{L-1} \sum\limits_{n=0}^{N_l-1} m^i_{l,n} x^i_{l,n} \leq M^i, \\ \sum\limits_{l=0}^{L-1} \sum\limits_{n=0}^{N_l-1} g^i_{l,n} x^i_{l,n} \leq G^i. \end{cases} \tag{2}$$

Similarly, as the amount of bandwidth used on a service node can not be more than the capacity, the bandwidth resource constraint is expressed as

TABLE 1
List of Symbols

Symbol	Definition
V	Set of service nodes in cloud computing.
E	Set of network links in cloud computing.
K	Maximum number of service nodes.
V_a	Set of service nodes deployed tasks.
K_a	Number of service nodes deployed tasks.
T	Set of tasks.
L	Number of tasks.
N_l	Number of sub-tasks for the lth task.
t_l	Set of sub-tasks for the lth task.
$t_{l,n}$	The nth sub-task for the lth task.
v_i	The ith service node.
C^i	Resource capacity for CPU on v_i.
M^i	Resource capacity for Memory on v_i.
G^i	Resource capacity for GPU on v_i.
B^i	Resource capacity for Bandwidth on v_i.
$c^i_{l,n}$	Resource requirement of $t_{l,n}$ for CPU on v_i.
$m^i_{l,n}$	Resource requirement of $t_{l,n}$ for Memory on v_i.
$g^i_{l,n}$	Resource requirement of $t_{l,n}$ for GPU on v_i.
$b^i_{l,n}$	Resource requirement of $t_{l,n}$ for Bandwidth on v_i.
$x^i_{l,n}$	Indicate whether $t_{l,n}$ is assigned on v_i.
$y^{i,j}_{l,n,\hat{n}}$	Indicate whether a bandwidth cost is available for $t_{l,n}, t_{l,\hat{n}}$.
$\tilde{b}^i_{l,n}$	Bandwidth cost for running $t_{l,n}$.
$U^s_{l,n}$	Set of successors of $t_{l,n}$.
$U^p_{l,n}$	Set of predecessors of $t_{l,n}$.
Z_1	Objective function of resource utilization.
Z_2	Objective function of bandwidth cost.
Z	Total objective function.
α, β	Weight values.

$$\sum_{l=0}^{L-1} \sum_{n=0}^{N_l-1} \tilde{b}^i_{l,n} x^i_{l,n} \leq B^i. \tag{3}$$

Furthermore, all the sub-tasks from the task set T are deployed to the service nodes in V and any sub-task can be allocated just only once. Thus, we can obtain the following:

$$\sum_{i=0}^{K-1} \sum_{l=0}^{L-1} \sum_{n=0}^{N_l-1} x^i_{l,n} = \sum_{l=0}^{L-1} N_l. \tag{4}$$

In this paper, one of our goals is to improve the resource utilization of cloud computing by reducing the number of service nodes used. For service node v_i, the resource utilization includes three parts: CPU utilization, memory utilization and GPU utilization, which are denoted by $\varphi^i_c, \varphi^i_m, \varphi^i_g$, respectively. They can be computed as follows:

$$\begin{cases} \varphi^i_c = \sum\limits_{l=0}^{L-1} \sum\limits_{n=0}^{N_l-1} c^i_{l,n} x^i_{l,n} / C^i, \\ \varphi^i_m = \sum\limits_{l=0}^{L-1} \sum\limits_{n=0}^{N_l-1} m^i_{l,n} x^i_{l,n} / M^i, \\ \varphi^i_g = \sum\limits_{l=0}^{L-1} \sum\limits_{n=0}^{N_l-1} g^i_{l,n} x^i_{l,n} / G^i. \end{cases} \tag{5}$$

According to the preferences of different resource types, we provide the weight values for CPU, memory and GPU as α_c, α_m and α_g, respectively. The resource utilization Z_1 for

our system model is obtained by a linear weighted sum method as follows:

$$Z_1(X) = \frac{1}{K_a} \sum_{i=0}^{K_a-1} \left(\alpha_c \varphi_c^i + \alpha_m \varphi_m^i + \alpha_g \varphi_g^i \right), \tag{6}$$

where $X = \left\{ x_{l,n}^i, \forall v_i \in V, \forall t_{l,n} \in T \right\}$ is a feasible solution for allocating the available resources of service nodes to the requested tasks in cloud computing. The parameter K_a $(K_a \leq K)$ denotes the number of service nodes that are used to deploy the requested tasks. In addition, $\alpha_c + \alpha_m + \alpha_g = 1$.

Another goal is to optimize the bandwidth cost by deploying the adjacent sub-tasks to the same service node. Let us denote the bandwidth cost as Z_2 which can be expressed as follows:

$$Z_2(X) = \frac{1}{K_a} \sum_{i=0}^{K_a-1} \sum_{l=0}^{L-1} \sum_{n=0}^{N_l-1} \tilde{b}_{l,n}^i x_{l,n}^i / B^i. \tag{7}$$

In cloud environment, we think that when a service node is running it will increase the operating cost [30], [31]. In order to reduce the operating cost, the number of service nodes that are used by the requested tasks should be as small as possible and the service nodes in idle mode can be in sleep or shutdown states [31]. In addition, the adjacent sub-tasks from a task are assigned on the same service node to save the bandwidth and delay costs, and improve the performance of cloud computing. We need to simultaneously optimize these two objectives which are $Z_1(X)$ and $Z_2(X)$. Specifically, our purpose is to seek an optimal solution which maximizes the resource utilization $Z_1(X)$ and minimizes $Z_2(X)$. Thus the combined objective $Z(X)$ which involves maximizing $Z_1(X)$ and minimizing $Z_2(X)$ can be expressed as

$$\text{maximize} \quad Z(X) = \beta_1 Z_1(X) + \beta_2 (1 - Z_2(X)), \tag{8}$$

where β_1 and β_2 are the weight values for $Z_1(X)$ and $1 - Z_2(X)$, respectively. The preferences of different objectives can be adjusted by varying β_1 and β_2, and we consider $\beta_1 + \beta_2 = 1$. However, this optimization problem is regarded as NP-hard problem, which means a high computational complexity order for finding an optimal solution. Meta-heuristic optimization algorithms are effective approaches to solve the combinatorial optimization problem, but the performance of solution quality and convergence time will degrade with the increase in the scale of the problem. Distributed optimization algorithms, which are based on decomposition and multi-agent systems, can improve the optimization solution effectively.

In the next section, we propose the HMAO algorithm, which is an optimization approach using hierarchical multi-agent framework, to address the resource allocation problem in cloud computing.

4 Hierarchical Multi-Agent Optimization

In this paper, we address a joint optimization problem of maximizing the resource utilization in cloud computing systems and reducing the bandwidth cost [32], [33]. To reduce the computational complexity of the optimization problem, we decompose the total objective into two optimization sub-problems: maximizing $Z_1(X)$ and minimizing $Z_2(X)$, and these two optimization sub-problems will be solved, accordingly.

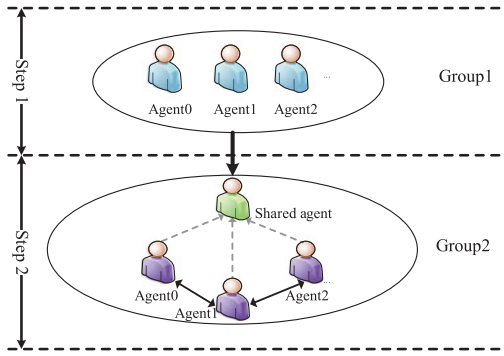


Fig. 3. Proposed hierarchical multi-agent framework.

First, an improved GA is introduced to find the optimal solution which maximizes $Z_1(X)$. All sub-tasks from T make up an ordered list as an individual that represents a feasible solution. We use a roulette wheel selection approach [34] to obtain those individuals with higher fitness values. Besides, two-point crossover and signal-point mutation are also used [35]. A set V_a of service nodes that are used to deploy the requested tasks can be provided as the optimal solution by the improved GA.

Then, a multi-agent optimization algorithm is proposed to solve the sub-problem of minimizing $Z_2(X)$. We use a shared agent to hold the information of resource allocation for the service nodes, and assign service agents to each service node for assisting in resource management [36]. Different service agents can cooperate, coordinate and compete with each other to optimize their objectives with respect to the behavior criteria [28]. To keep the diversity of feasible solutions and avoid the occurrence of premature convergence, both selection and exchange operators [36] are achieved by a probabilistic method. In addition, a feasible solution consists of all the sub-tasks from the available service agents and those sub-tasks on a service agent are considered as part of a feasible solution. We can optimize the objectives of the service agents with cooperative co-evolutionary method to obtain a global optimal solution [23].

Fig. 3 shows the proposed hierarchical multi-agent framework. The procedure for addressing the optimization problem is divided into two steps. In the first step, the improved GA is used to find the optimal solution which maximizes $Z_1(X)$, and each agent represents an individual. In the second step, the MAO algorithm is applied to seek the optimal solution which minimizes $Z_2(X)$. Shared agent can hold the information concerning task deployment and resource allocation for all service nodes, and support service agents to access and update the information about task deployment and resource allocation in real-time. Service agents are assigned on each service node and provide help for resource management. Service agents can obtain the information about task deployment and resource allocation by accessing the shared agent, and swap their sub-tasks with each other to improve the resource allocation solution.

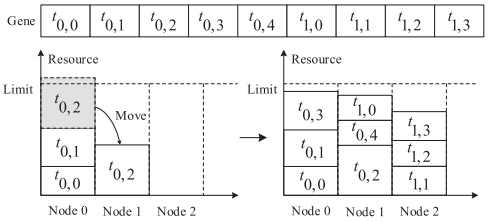


Fig. 4. Procedure for decoding an individual.

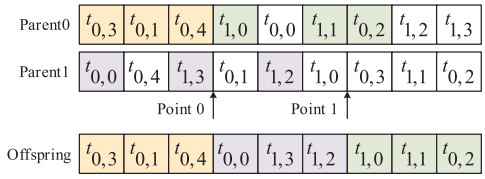


Fig. 5. Example of two-point crossover operator.

4.1 Improved GA

For the sub-problem of optimizing the resource utilization, the objective is to maximize $Z_1(X)$ and the constraints need to be satisfied. The improved GA is introduced to solve this optimization problem. The improvement consists of two parts: procedure of decoding an individual and initial population generation. We implement two kinds of procedure for decoding an individual by index priority and first-fit rule [37], [38], respectively. One decoding procedure based on index priority is used for the off-line resource allocation and the other procedure of decoding by first-fit rule is applied for the on-line resource allocation. Furthermore, for the proposed HMAO algorithm in on-line resource allocation, in order to improve the quality of the initial solutions, an individual is encoded by the inter-dependent relationships of those sub-tasks in sequence and the rest of the individuals are randomly encoded to keep diversity of feasible solutions.

In the improved GA, a population includes P individuals, an individual $p \in P$ is encoded by a permutation representation [35], [39] and consists of $\sum_{l=0}^{L-1} N_l$ genes in order, where a gene represents a sub-task. During the process of decoding, all service nodes in V are sorted in ascending order by index, and the service nodes with low index have high priority to be deployed with the requested tasks. Thus, a sub-task is deployed depending upon the priority of a service node until the resource requirements are satisfied. By the decoding method, the sub-tasks from an individual can be allocated to these service nodes in V in sequence, and the result of resource allocation indicates a feasible solution.

Fig. 4 illustrates the procedure for decoding an individual. There are two tasks: t_0 with 5 sub-tasks and t_1 with 4 sub-tasks, an individual is expressed as $p = \{t_{0,0}, t_{0,1}, t_{0,2}, t_{0,3}, t_{0,4}, t_{1,0}, t_{1,1}, t_{1,2}, t_{1,3}\}$. Three service nodes are denoted as v_0, v_1 and v_2, respectively, and their resource capacities are limited. First, it is seen that sub-tasks $t_{0,0}$ and $t_{0,1}$ are allocated to service node v_0, but the required resources for v_0 are more than the capacities after deploying $t_{0,2}$ to v_0. As a result, sub-task $t_{0,2}$ is moved to service node v_1. Due to the high priority of service nodes with low index, sub-task $t_{0,3}$ will be placed to service node v_0. Similarly, we deploy sub-tasks $t_{0,4}$ and $t_{1,0}$ to service node v_1, sub-tasks $t_{1,1}, t_{1,2}$ and $t_{1,3}$ to service node v_2.

Three operators for selection, crossover and mutation are used as follows:

- *Selection operator:* A roulette wheel selection [34] is applied to obtain the individuals with high fitness values. All individuals are sorted in ascending order by their fitness values and the cumulative probability distribution function (CDF) is computed accordingly. Then we choose the candidates through the concept of the survival of the fittest,which means an individual with high fitness is more likely to be chosen.
- *Crossover operator:* Two-point crossover [35] is used as the crossover operator and executed with crossover probability p_c. We randomly select two gene points for two individuals (one each from the mother and father) in the field $\left\{1, \sum_{l=0}^{L-1} N_l - 2\right\}$ to mate with each other, each offspring inherits some of the genes from their parents, respectively. An example of two-point crossover is described in Fig. 5. Two individuals $p_0 = \{t_{0,3}, t_{0,1}, t_{0,4}, t_{1,0}, t_{0,0}, t_{1,1}, t_{0,2}, t_{1,2}, t_{1,3}\}, p_1 = \{t_{0,0}, t_{0,4}, t_{1,3}, t_{0,1}, \quad t_{1,2}, t_{1,0}, t_{0,3}, t_{1,1}, t_{0,2}\}$ are given as the parents, and two gene points are randomly generated, such as 3 and 6. First, an offspring inherits $t_{0,3}, t_{0,1}$ and $t_{0,4}$ from $parent0$ as the genes of itself. Then 3 genes ($point1 - point0 = 3$) from $parent1$ need to be searched from low to high, and they can not be the same as that of the offspring. Therefore, genes $t_{0,0}, t_{1,3}$ and $t_{1,2}$ from $parent1$ are inherited as the genes of the offspring. In the same way, the offspring can inherit genes $t_{1,0}, t_{1,1}$ and $t_{0,2}$ from $parent0$. Thus the offspring is indicated as $\{t_{0,3}, t_{0,1}, t_{0,4}, t_{0,0}, t_{1,3}, t_{1,2}, t_{1,0}, t_{1,1}, t_{0,2}\}$. Similarly, $\{t_{0,0}, t_{0,4}, t_{1,3}, t_{0,3}, t_{0,1}, t_{1,0}, t_{1,2}, t_{1,1}, t_{0,2}\}$ can represent another offspring.
- *Mutation operator:* Mutation operator [35] is carried out with mutation probability p_m, where we randomly choose two gene points from an individual and exchange their genes to gemerate a new individual.

The improved GA is described in Algorithm 1. Let us denote the maximum number of iterations as $Iter_{max}$. At the beginning, the initial population P is randomly produced, we compute the fitness of the objective function for those individuals from population P and obtain the CDF. Then the population of offsprings can be obtained by running selection, crossover and mutation operators, respectively. The optimal solution of maximizing $Z_1(X)$ will be given in iterative evolution. The termination criterion is met when the number of iterations is greater than $Iter_{max}$.

According to the improved GA, we obtain an optimal solution which maximizes the resource utilization, where the optimal solution indicates a set V_a of service nodes that are used to deploy the requested tasks and contains K_a service nodes. However, the bandwidth cost and load-balancing are not considered for this sub-problem. The following Section 4.2 will discuss the sub-problem of minimizing the bandwidth cost and load-balancing based on a set V_a of service nodes.

Algorithm 1. Improved GA

Initialize: Population P, individual $p \in P$, crossover probability p_c, mutation probability p_m, maximum number of iterations $Iter_{max}$;
for $Iter = 0$ to $Iter_{max}$ **do**
  Compute the fitness for all individuals P;
  Obtain the CDF based on the fitness values;
  Run selection operator to get P candidates;
  for $i = 0$ to $\frac{length(P)}{2}$ **do**
    Randomly choose two individuals;
    Generate a random number p_r;
    if $p_r \leq p_c$ **then**
      Run two-point crossover operator;
    end if
  end for
  Produce offspring population;
  for $i = 0$ to $length(P)$ **do**
    Randomly choose an individual;
    Generate a random number p_r;
    if $p_r \leq p_m$ **then**
      Run mutation operator;
    end if
  end for
  Update population P;
end for

4.2 Multi-Agent Optimization

For set V_a of service nodes, we propose a multi-agent optimization approach to address the optimization sub-problem of minimizing $Z_2(X)$ with the constraints. For the MAO, the agents have four characteristics: environment-aware, autonomy, social behavior and load-balancing [29], which are described in detail as follows:

1) *Environment-aware:* The environment in the MAO algorithm consists of all the agents and their relationships, where the agents are designed as a shared agent and K_a service agents in advance, and we use an organized architecture to illustrate their relationships. A service agent can obtain the information about resource allocation by accessing the shared agent, and interact with other service agents to better adapt to the current environment condition, that is, to achieve a higher fitness value. A shared agent provides access services for the service agents in order to help with resource management.
2) *Autonomy:* When the environment conditions are changed, an agent can autonomously make a decision for the next actions to adjust its fitness value by a set of actions and the behavior criteria. For a service agent, its action set includes four parts: access, selection, exchange and update. First, it obtains the information about resource allocation by accessing the shared agent. Then selection and exchange operators, which are described later in this section, can be executed, respectively. The results of interaction on service agents will be updated on the shared agent. In addition, the action set of a shared agent can provide three services which are storing, accessing and updating the information.
3) *Social behavior:* According to social behavior, the agents share the information about resource allocation, and the sub-tasks deployed on different service agents can be exchanged with each other. There are two kinds of social behavior which are between shared agent and service agents, and service agents with each other. For the former, the agents can share the information about resource allocation with all the service agents. For the latter, the aim is to exchange those sub-tasks that are deployed on different service agents to improve their objectives.
4) *Load-balancing:* Load-balancing is an important issue to ensure that the service nodes are being used sufficiently. Therefore, considering load-balancing, a service agent runs the exchange operator by cooperating, coordinating and competing with other service agents.

In order to better describe the MAO algorithm, some important specifications are given as follows:

- *Shared agent:* Shared agent, which is denoted by a_s, holds the information about task deployment and resource allocation for all service agents, it can support the service agents to access and update the information in real-time.
- *Service agent:* Service agents are assigned to the service nodes in V_a to assist in resource management, where they can access and update the information about resource allocation on the shared agent. Moreover, the sub-tasks deployed on different service agents can be migrated and swapped with each other by selection and exchange operators. The service agent assigned to v_i is indicated as a_i and all the service agents make up a set A_a of the service agents.
- *Adjacent agent:* We assume that $\forall t_{l,n} \in a_i$, $\exists t_{l,\hat{n}} \in a_j, i \neq j$, where $t_{l,\hat{n}} \in U^p_{l,n} \cup U^s_{l,n}$, then a_i, a_j are seen as adjacent agents.
- *Active sub-task:* For a sub-task $\forall t_{l,n} \in a_i$, its adjacent sub-tasks are migrated and swapped to a_i to improve the bandwidth cost by selection and exchange operators. Sub-task $t_{l,n}$ is defined as an active sub-task.
- *Host service agent:* If an active sub-task $t_{l,n} \in a_i$, the service agent a_i is considered as the host service agent.
- *Feasible solution:* All sub-tasks placed on service agent a_i are indicated as Λ_i, where Λ_i is considered as part of a feasible solution. Therefore, a feasible solution can be expressed as $X = \bigcup_{i=0}^{K_a-1} \Lambda_i$.
- *Objective function:* A feasible solution is decomposed into K_a parts, thus the objective can be re-written as: $Z_2(X) = \frac{1}{K_a} \sum_{i=0}^{K_a-1} Z_2(\Lambda_i)$.

In the MAO algorithm, there is a shared agent a_s and K_a service agents, the agents can communicate with each other by network links, such as a_s and a_i, and a_i and $a_j, i \neq j$. The service agents can share the information about resource allocation by accessing the shared agent a_s, migrate and swap their sub-tasks with each other to reduce their bandwidth costs. To keep the diversity of feasible solutions, selection and exchange operators are implemented based on a probabilistic method which is discussed later in this section.

Generally, the procedure of the MAO includes four parts: obtain the adjacent agents, choose source and target

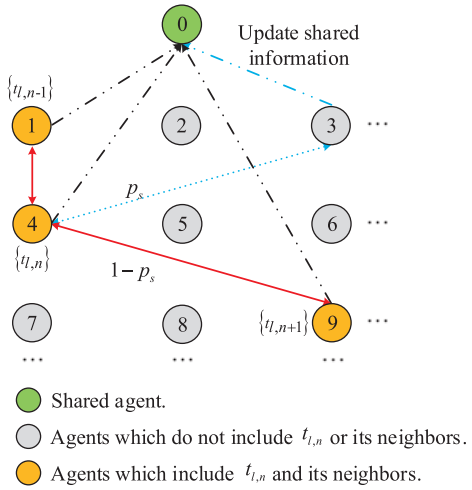


Fig. 6. Architecture for multi-agent optimization.

sub-tasks, execute the exchange operator, and update the shared information. First, let us select sub-task $\forall t_{l,n} \in a_i$ as the active sub-task, all its adjacent sub-tasks $U_{l,n} = U^p_{l,n} \cup U^s_{l,n}$ and their service agents can be obtained by accessing the shared agent. Then selection and exchange operators can be carried out for each sub-task from corresponding $U_{l,n}$. For the selection operator, we need to choose the source and target sub-tasks, where the target sub-task is selected by a probabilistic method, and the source sub-task from a_i is obtained by a priority-based selection mechanism. The priority-based selection mechanism consists of two parts, where one is the dependence relationships of sub-tasks for two adjacent service agents and the other is the used resources of server nodes that are illustrated in detail in Section 4.2.2. For exchange operator, we can migrate and swap the source and target sub-tasks among different service agents with multiple constraints, which include objective optimization, resource constraints, load-balancing and diversity of feasible solutions that are clearly described in Section 4.2.3. Furthermore, the exchange operation can be carried out with exchange probability p_e if the objective fitness is not improved at this instance. When the exchange is finished, the shared information will be updated.

An example of the MAO algorithm is illustrated in Fig. 6. We assume that a host service agent is a_4 and an active sub-task is $t_{l,n}$. The service agent a_4 can obtain the information about the adjacent sub-tasks for $t_{l,n}$ by visiting the shared agent, and there are two adjacent sub-tasks $t_{l,n-1} \in a_1, t_{l,n+1} \in a_9$. For sub-task $t_{l,n+1}$, it is chosen as the target sub-task with selection probability $1 - p_s$ or else a random sub-task $t_{\hat{l},\hat{n}} \in a_3$ is chosen to be the target sub-task. A source sub-task from a_4 is selected by a priority-based selection mechanism. Then we can run the exchange operation and update the shared information. Similarly, we can proceed with the processing for $t_{l,n-1}$.

Next, we discuss three main parts for the MAO algorithm.

4.2.1 Target Selection

In the MAO algorithm, to optimize the bandwidth cost, our purpose is to assign the adjacent sub-tasks from a task to the same service agent as far as possible. In addition, a probabilistic method is used to select the target sub-task to avoid premature convergence.

For active sub-task $t_{l,n} \in a_i$, the set of its adjacent sub-tasks is $U_{l,n}$, and we assume that a candidate adjacent sub-task is $t_{l,\hat{n}} \in U_{l,n}, t_{l,\hat{n}} \in a_j, i \neq j$ and a random sub-task is denoted as $t_{\hat{l},\hat{n}} \in a_k, i \neq k$. Therefore, the target sub-task is $t_{\hat{l},\hat{n}}$ with probability p_s, otherwise $t_{l,\hat{n}}$. The algorithm for target selection is described in Algorithm 2.

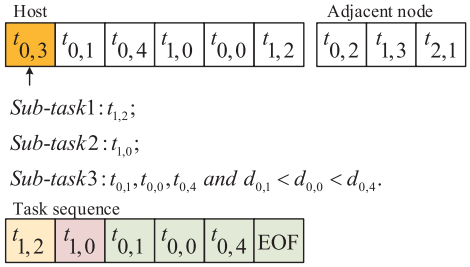


Fig. 7. Source sub-task candidate list.

4.2.2 Source Selection

Considering load-balancing and objective optimization, a priority-based source sub-task selection mechanism is proposed. In the host service agent, all available sub-tasks for active sub-task $t_{l,n} \in a_i$ are sorted in descend order by the priority, which includes two categories: the dependence relationships of sub-tasks for the host and adjacent service agents, and the used resources. There are three sub-task types according to the dependence relationships as follows:

- *Sub-task1:* Let us denote active sub-task as $t_{l,n} \in a_i$, an adjacent service agent as $a_j, i \neq j$ and a set of available sub-tasks as $W_{l,n}$. If $t_{l',n'} \in W_{l,n}, t'_{l',n'} \in U_{l',n'}$, make $\exists t'_{l',n'} \notin W_{l,n}$ and $t'_{l',n'} \in a_j$, let $t_{l',n'}$ be sub-task1.
- *Sub-task2:* If $t_{l',n'} \in W_{l,n}, t'_{l',n'} \in U_{l',n'}$, make $\forall t'_{l',n'} \notin W_{l,n}$ and $t'_{l',n'} \notin a_j$, let $t_{l',n'}$ be sub-task2.
- *Sub-task3:* If $t_{l',n'} \in W_{l,n}, t'_{l',n'} \in U_{l',n'}$, make $\exists t'_{l',n'} \in W_{l,n}$, let $t_{l',n'}$ be sub-task3.

The precedence relations for three sub-task types are ranked as: sub-task1 $>$ sub-task2 $>$ sub-task3. Furthermore, for the same sub-task type, we calculate the resource utilization difference between a_i and the average value of the system for each candidate sub-task. All the candidate sub-tasks are sorted in ascending order by the differences.

Algorithm 2. Target Selection

1: **Initialize:** Probability p_s;
2: For $\forall t_{l,n} \in a_i, \exists t_{l,\hat{n}} \in U_{l,n}, t_{l,\hat{n}} \in a_j, a_i, a_j \in A_a, i \neq j$, $t_{\hat{l},\hat{n}} \in a_k$, $i \neq k$;
3: Generate a random number p_r;
4: **if** $p_r \geq p_s$ **then**
5: Let $t_{l,\hat{n}}$ be the target sub-task;
6: **else**
7: Let $t_{\hat{l},\hat{n}}$ be the target sub-task;
8: **end if**

Fig. 7 describes the procedure of a source sub-task candidate list. Host service agent is indicated as $\{t_{0,3}, t_{0,1}, t_{0,4}, t_{1,0}, t_{0,0}, t_{1,2}\}$, active sub-task is $t_{0,3}$ and an adjacent service agent consists of $t_{0,2}, t_{1,3}$ and $t_{2,1}$. We can find

that $t_{1,2}$ is sub-task1, $t_{1,0}$ is sub-task2, $t_{0,1}, t_{0,0}$ and $t_{0,4}$ are sub-task3. For sub-task3, the differences of the sub-tasks are ranked as $d_{0,1} < d_{0,0} < d_{0,4}$. As a result, the task list can be indicated as $\{t_{1,2}, t_{1,0}, t_{0,1}, t_{0,0}, t_{0,4}\}$.

For the sub-task candidate list, the source sub-task can be chosen by the priority, and different source sub-tasks have an influence on the performance. The algorithm for source selection is shown in Algorithm 3.

Algorithm 3. Source Selection

For active sub-task $t_{l,n} \in a_i$, an adjacent service agent $a_j, i \neq j$;
Make a set of available sub-tasks $W_{l,n}$;
for $\forall t_{l',n'} \in W_{l,n}$ **do**
  Obtain the set of its adjacent sub-tasks $U_{l',n'}$;
  For $t'_{l',n'} \in U_{l',n'}$;
  if $\exists t'_{l',n'} \notin W_{l,n}$ and $t'_{l',n'} \in a_j$ **then**
    Let $t_{l',n'}$ be sub-task1;
  else if $\forall t'_{l',n'} \notin W_{l,n}$ and $t'_{l',n'} \notin a_j$ **then**
    Let $t_{l',n'}$ be sub-task2;
  else
    Let $t_{l',n'}$ be sub-task3;
  end if
  Assume $t_{l',n'}$ is a source sub-task and calculate the difference between the resource utilization of a_i and the average value of the system;
end for
Sort the sub-task candidate list by precedence.

4.2.3 Exchange Procedure

For active sub-task $t_{l,n}$, the target sub-task and source sub-task candidate list are given by target and source selection operations, then the service agents can cooperate, coordinate and compete with each other to migrate and swap their sub-tasks to improve the objectives through the exchange operator. In this context, there are four situations to be considered as follows:

- *Objective optimization:* For the exchange procedure, our aim is to migrate and swap these sub-tasks on different service agents to reduce their bandwidth costs. That is, the objective fitness should be improved for the host service agent after running the exchange operator.
- *Resource constraints:* The number of resources used for each service agent can not exceed its resource capacities.
- *Load-balancing:* Load-balancing is implemented by a priority-based source sub-task selection mechanism and the procedure of migrating and swapping the sub-tasks.
- *Diversity of feasible solutions:* Exchange operator is achieved by a probabilistic method, when the objective optimization is not satisfied, the exchange procedure can continue to be carried out with a lower probability.

Next, we describe the exchange procedure in detail.

Let us denote the source sub-task list by $Q_{l,n}$. For $\forall t_{l',n'} \in Q_{l,n}$, we first ensure that the objective result is improved by the exchange operation. If it is not, the exchange procedure can keep running with exchange probability p_e. Considering the load-balancing for all the service agents, if the load-balancing constraint in Equation (9) is satisfied, we will just migrate the target sub-task from a_j to a_i and there will be nothing to do for the source sub-task. Besides, the migration will be also considered when there is no available source sub-task in the candidate list. The load-balancing constraint can be expressed as

$$\eta_i < \eta_j + 2 * \bar{\eta}, \tag{9}$$

where η_i and η_j show the resource utilization of a_i and a_j after the migration, respectively. The parameter $\bar{\eta}$ indicates the average resource utilization for all sub-task types. In most cases, we need to swap the target and source sub-tasks with each other between a_i and a_j. Note that the migration and exchange are satisfied with the resource constraints. The exchange procedure is shown in Algorithm 4.

Algorithm 4. Exchange Procedure

For target sub-task $t_{\hat{l},\hat{n}}$, source sub-task candidate list $Q_{l,n}$, exchange probability p_e;
for $\forall t_{l',n'} \in Q_{l,n}$ **do**
  if The objective optimization is invalid **then**
    Generate a random number p_r;
    if $p_r > p_e$ **then**
      Continue;
    end if
  end if
  if ($t_{l',n'} = EOF$ or ($t_{l',n'} \neq EOF$ and hold Equation (9))) and meet with resource constraints **then**
    Run the migration and break;
  else if $t_{l',n'} \neq EOF$ and hold resource constraints **then**
    Run the exchange and break;
  end if
end for
Update the shared information on the shared agent.

Algorithm 5. MAO Algorithm

Input: Maximum number of iterations M, selection probability p_s, exchange probability p_e;
for $m = 0$ to M **do**
  for $\forall a_i \in A_a$ **do**
    for $\forall t_{l,n} \in a_i$ **do**
      Obtain $U_{l,n}$ by visiting a_s;
      for $\forall t_{l,\hat{n}} \in U_{l,n}, t_{l,\hat{n}} \in a_j, i \neq j$ **do**
        Run the target selection operation;
        Run the source selection operation;
        Run the exchange procedure;
      end for
    end for
    Update Λ_i for a_i;
  end for
  Get the feasible solution $X = \bigcup_{i=0}^{K_a-1} \Lambda_i$ and compute the objective fitness value;
end for
Obtain an approximate optimal solution.

The MAO algorithm is an iterative optimization algorithm based on multi-agent systems and Algorithm 5 describes the entire MAO algorithm. We assume that the

maximum number of iterations is M. During an iterative evolution, all the sub-tasks from each service agent run the selection and exchange operations, and the objective value for this service agent can be improved with high probability. All the service agents can work together with cooperative co-evolutionary method. Thus, the global optimal solution can be found by increasing the number of iterations.

4.3 HMAO Algorithm in On-Line Resource Allocation

The proposed HMAO algorithm is easily to be applied to the on-line resource allocation by a few modifications. In this paper, we consider the scenario of allocating the available resources related to service nodes to the requested tasks in batch mode. That is, the requested tasks need to be performed are collected and will be handled at a fixed time slot.

All the available resources related to service nodes make up an available resource pool. In addition, we assume that some new requested tasks appear and several old requested tasks are over in each time slot. Before running the proposed HMAO algorithm at every time, we need to check whether there are several completed old requested tasks in the last time slot. The resources used by these completed old requested tasks can free and put into the resource pool to be available for deploying the coming requested tasks in the current time slot. Then we can run the proposed HMAO algorithm to assign the available resources in the resource pool to the new requested tasks.

Algorithm 6. HMAO Algorithm in On-Line Resource Allocation

1: **Input at time** t**:** Set $T_{new,t} \in T$ of the tasks coming at t, set $T_{old,t-1} \in T$ of the tasks ending at $t-1$;
2: Release the required resources of the old tasks in $T_{old,t-1}$ and update the resource information for the service nodes;
3: Generate the initial population P for $T_{new,t}$;
4: Run the GA and obtain the set $V_{a,t}$ of service nodes that are used to deploy the new tasks;
5: Carry out the MAO algorithm for $V_{a,t}$;
6: Obtain an approximate optimal solution of allocating the available resources to $T_{n,t}$.
7: Time: $t \leftarrow t+1$.

In order to improve the quality of the initial solutions for the GA, an individual is encoded by the inter-dependent relationships of those sub-tasks in sequence and the rest of the population are randomly encoded to keep diversity of feasible solutions. For the decoding procedure of the GA, we sort the service nodes that are used to deploy the requested tasks in ascending order by the resource utilization. All the sub-tasks for an individual are assigned to the available service nodes with the first-fit rule [37], [38] and a new service node would be activated when any of the available service nodes can not meet the resource requirements of the sub-task to be assigned. Algorithm 6 describes the proposed HMAO algorithm in on-line resource allocation. For time slot t, let us denote the set of the new requested tasks as $T_{new,t} \in T$ and the set of the old requested tasks that are ending as $T_{old,t-1} \in T$. First, we end the old tasks from $T_{old,t-1}$ and release the required resources used. Then with existing resources used and physical resource constrains, we use the proposed HMAO algorithm to allocate the available resources of service nodes to the new requested tasks.

TABLE 2
Resource Requirements for Tasks and Service Nodes

Name	CPU (MHz)	Memory (GB)	GPU	Bandwidth (Mbps)
Network receiving	290	9.6	0	100
Capture	319	11.52	1	97
Tracking	435	12.48	1	95
Synchronization	638	12.48	1	92
Decoding	145	4.8	1	90
Server node	2900	96	8	1000

5 Performance Evaluation

In this section, we make the experiments for different number of communication tasks to verify the performance of the proposed HMAO algorithm with computer simulation results. Meanwhile, the performance of the proposed HMAO algorithm is analyzed by comparing with two existing baseline algorithms, which are GA and NSGA-II. The experimental platform is a high performance server, which is i7-4790k CPU, 16 GB memory and windows 10. Each communication task contains 5 parts: network receiving, capture, tracking, synchronization and decoding. All service nodes are homogeneous and have the same resource capability. The resource requirements for tasks and service nodes can be observed in Table 2. Note that the limitation of network links is not considered in this paper, such as switches and routers. Equation (8) is considered for performance metrics comparison between the proposed HMAO algorithm and other existing baseline algorithms.

5.1 Simulation Parameters Setup

First, we assume that the weights in Equation (8) have the same values, and can be described as $\alpha_c = \alpha_m = \alpha_g = \frac{1}{3}$ and $\beta_1 = \beta_2 = \frac{1}{2}$. The parameters for the improved GA are $P = 16, Iter_{max} = 5, p_c = 1.0, p_m = 0.1$. In the MAO algorithm, there are three main parameters: selection probability p_s, exchange probability p_e and the number of iterations, which have impact on the performance results. In order to better estimate this impact on the performance for different combinations of parameters, the Taguchi method of design-of-experiment (DOE) is used to generate the test cases and analyze the results of our experiments [29]. There are 3 factors and each factor contains 4 levels, and the combinations of different parameters for Taguchi method are shown in Table 3. Moreover, we develop the orthogonal table $L_{16}(4^3)$, where there are 16 cases and each case is carried out 10 times for 8, 16, 24, 32 tasks, respectively, to obtain the average results. Table 4 provides the orthogonal table $L_{16}(4^3)$ and the values related to the approximate solutions for different tasks.

Fig. 8 shows the main effects plot for means with different tasks. It can be obvious that the objective values are improved as p_s, p_e and the number of iterations increase.

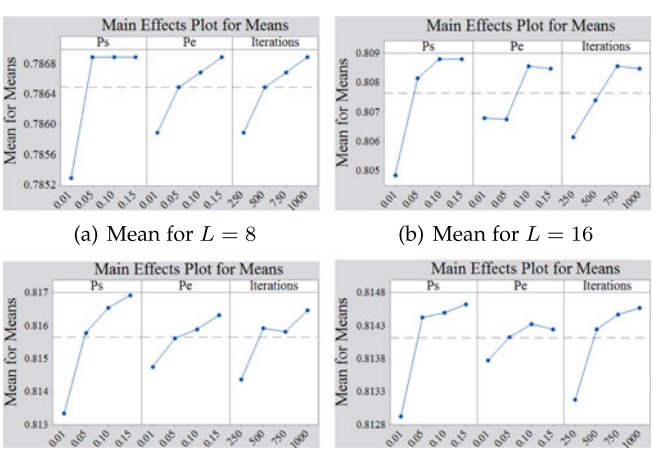
TABLE 3
Parameters for Taguchi Method

Factor	Level			
	1	2	3	4
p_s	0.0s1	0.05	0.10	0.15
p_e	0.01	0.05	0.10	0.15
M	250	500	750	1000

TABLE 4
Orthogonal Table $L_{16}(4^3)$ and the Optimal Solutions for $Z(X)$

No.	Factor			$Z(X)$			
	p_s	P_e	M	$L=8$	$L=16$	$L=24$	$L=32$
0	0.01	0.01	250	0.7829	0.8008	0.8103	0.8109
1	0.01	0.05	500	0.7853	0.8032	0.8141	0.8133
2	0.01	0.10	750	0.7861	0.8079	0.8138	0.8136
3	0.01	0.15	1000	0.7869	0.8075	0.8152	0.8139
4	0.05	0.01	500	0.7869	0.8088	0.8156	0.8146
5	0.05	0.05	250	0.7869	0.8062	0.8144	0.8136
6	0.05	0.10	1000	0.7869	0.8088	0.8167	0.8148
7	0.05	0.15	750	0.7869	0.8088	0.8164	0.8147
8	0.10	0.01	750	0.7869	0.8088	0.8161	0.8148
9	0.10	0.05	1000	0.7869	0.8088	0.8170	0.8148
10	0.10	0.10	250	0.7869	0.8088	0.8161	0.8141
11	0.10	0.15	500	0.7869	0.8088	0.8170	0.8143
12	0.15	0.01	1000	0.7869	0.8088	0.8170	0.8148
13	0.15	0.05	750	0.7869	0.8088	0.8170	0.8148
14	0.15	0.10	500	0.7869	0.8088	0.8170	0.8148
15	0.15	0.15	250	0.7869	0.8088	0.8167	0.8141

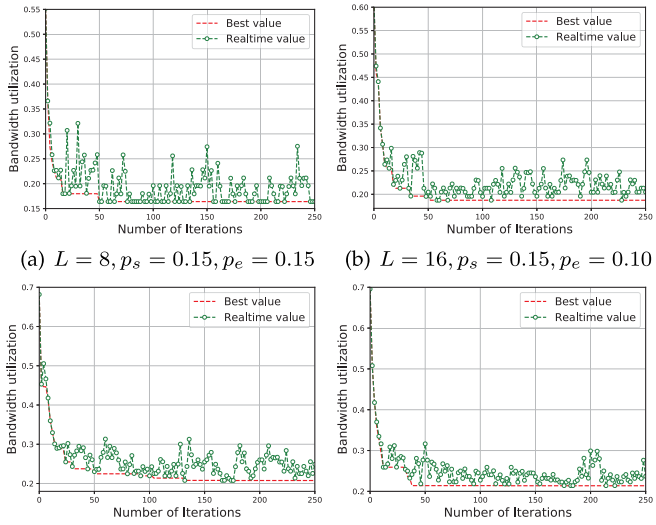
For the proposed HMAO algorithm, we can adjust the values of p_s, p_e to the diversity of feasible solutions, however, it will cost more time to seek an approximate solution for $Z(X)$. The ideal value of p_s for 8, 16, 24, 32 tasks is 0.15, the ideal value of p_e is 0.1 for 8, 24 tasks and 0.15 for 16, 32 tasks.



(a) Mean for $L = 8$ (b) Mean for $L = 16$

(c) Mean for $L = 24$ (d) Mean for $L = 32$

Fig. 8. Mean for $L = 8, 16, 24, 32$.



(a) $L = 8, p_s = 0.15, p_e = 0.15$ (b) $L = 16, p_s = 0.15, p_e = 0.10$

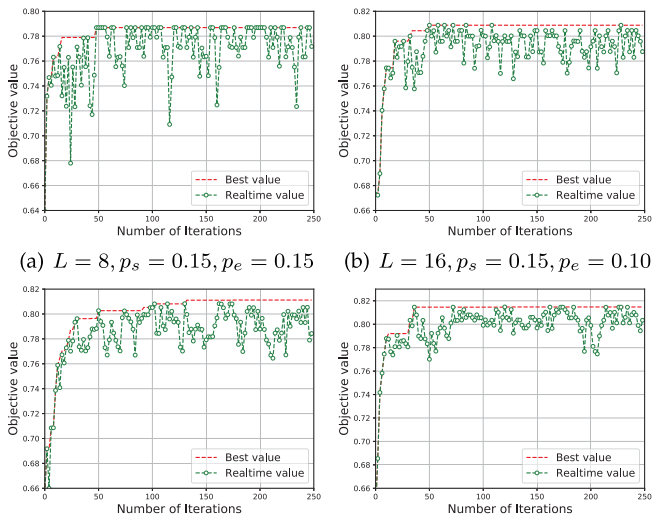
(c) $L = 24, p_s = 0.15, p_e = 0.15$ (d) $L = 32, p_s = 0.15, p_e = 0.10$

Fig. 9. Bandwidth utilization for $L = 8, 16, 24, 32$.

5.2 Numerical Simulation

Based on the parameters above, the maximum number of iterations is set as 250, we make four different experiments, where the number of requested tasks is 8, 16, 24 and 32, respectively, to evaluate convergence of the proposed HMAO algorithm. As it is shown in Fig. 9, we illustrate the relationships between iterations and the bandwidth utilization for different tasks. Figs. 9a and 9c indicate the procedure for seeking the optimal solution of bandwidth utilization for $L = 8, 24$ with $p_s = 0.15$, $p_e = 0.15$, and the best values 0.1640 and 0.0.2076 can be observed at 49 and 176 iterations, respectively. For $L = 16, 32$, $p_s = 0.15$, $p_e = 0.10$, their results for optimizing the bandwidth utilization are depicted in Figs. 9b and 9d, and we can obtain the optimal solution results 0.1872 and 0.2137 at 73 and 121 iterations, respectively.

Similarly, Fig. 10 describes the evolutionary plots of the iterative optimal solutions for different tasks, which include the best and real-time objective values. Figs. 10a and 10c



(a) $L = 8, p_s = 0.15, p_e = 0.15$ (b) $L = 16, p_s = 0.15, p_e = 0.10$

(c) $L = 24, p_s = 0.15, p_e = 0.15$ (d) $L = 32, p_s = 0.15, p_e = 0.10$

Fig. 10. Objective values for $L = 8, 16, 24, 32$.

TABLE 5
Results of Simulation for the HMAO Algorithm

L	M	Min	Max	Mean	Std	Time(min)
8	250	0.7869	0.7869	0.7869	0.0	0.2375
16		0.8046	0.8088	0.8084	0.0013	0.4777
24		0.8141	0.8170	0.8164	0.0012	0.7205
32		0.8104	0.8148	0.8137	0.0015	0.9703
8	500	0.7869	0.7869	0.7869	0.0	0.4995
16		0.8088	0.8088	0.8088	0.0	1.0125
24		0.8170	0.8170	0.8170	0.0	1.5177
32		0.8146	0.8148	0.8147	0.0001	2.0164

show the simulation results of 8, 24 tasks with $p_s = 0.15$, $p_e = 0.15$, Figs. 10b and 10d show the simulation results of 16, 32 tasks with $p_s = 0.15$, $p_e = 0.10$. It can be observed that the objective values for 8, 16, 24 and 32 tasks converge to the approximate solution results as 0.7869, 0.8088, 0.8112 and 0.8148 at 49, 73, 176 and 121 iterations, respectively. Therefore, it can be depicted that the proposed HMAO algorithm is an effective optimization algorithm to solve the problem of resource allocation in cloud computing and has a better convergence performance.

As the number of iterations have impact on the performance of the proposed HMAO algorithm, therefore, the experiments with 250 and 500 iterations are carried out for $L = 8, 16, 24$ and 32, respectively. Each case runs 10 times, we can obtain minimum, maximum, mean and standard deviation of the optimal solutions and the average computing time, which are shown in Table 5. It can be observed from Table 5 that the performance of the proposed HMAO algorithm is high when the number of iterations increases, i.e., the means for 8, 16, 24 and 32 tasks with 250 iterations are 0.7869, 0.8084, 0.8164 and 0.8137, respectively, while the means are 0.7869, 0.8088, 0.8170 and 0.8147 with 500 iterations, respectively. That is, the potential optimal solution for $Z(X)$ is more likely to be sought by exploring and exploiting the solution space iteratively. Besides, the results indicate that the computing time of the proposed HMAO algorithm increases almost linearly with the increase in the number of iterations.

5.3 Performance Comparison With the Baseline Algorithms

In order to further discuss the effectiveness of the proposed HMAO algorithm, we compare the proposed HMAO algorithm for different tasks with two existing algorithms, which are GA and NSGA-II.

GA. The genetic algorithm described in [9] is used in our paper. We randomly select two individuals from the population P to be the parents, and they can mate with each other by the two-point crossover operator with crossover probability p_c to generate their offsprings. If the fitness value of an offspring is superior to its parent, we will consider it as a candidate in the next generation. Otherwise, the mutation operator is carried out with mutation probability p_m. An individual that has a better fitness value between the offspring and its parent will be seen as a new individual in the next generation.

NSGA-II. We introduce NSGA-II in [15] to address our problem, two optimization sub-problems are $Z_1(X)$ and $1 - Z_2(X)$, respectively. A binary tournament selection method [34] is applied to make decision for choosing the parents from the population P to mate with each other. In crossover operator, we randomly select two gene points of one individual to exchange their genes equivalently with the other individual under crossover probability p_c. A bitwise mutation is executed with mutation probability p_m.

TABLE 6
Parameter Setting for HMAO, GA, and NSGA-II

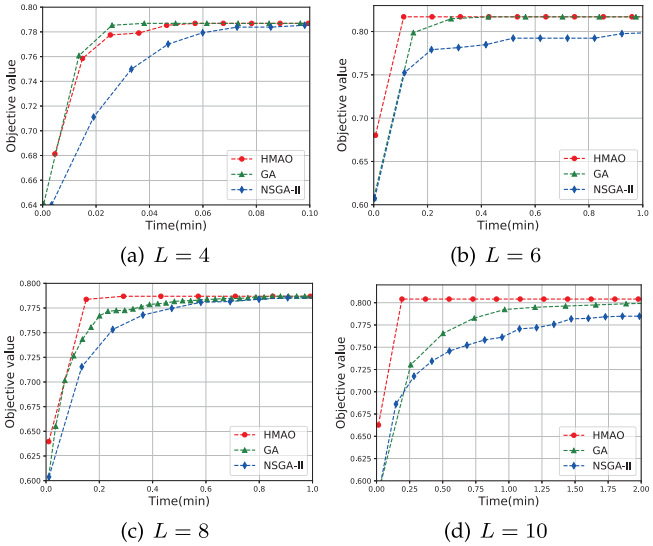
Parameters	Value
Number of tasks L	$\{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
Number of service nodes K	$\{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
Maximum iterations M	HMAO: 1000; GA,NSGA-II: 5000
Population size P	HMAO: 16; GA,NSGA-II: 100
Selective probability p_e	0.15
Exchange probability p_s	0.15
Crossover probability p_c	1.0
Mutation probability p_m	0.1
Running time	10

With the above analysis of the proposed HMAO algorithm, we set the maximum iterations as 1,000 for the proposed HMAO algorithm, 5,000 for GA and NSGA-II, the population P is set as 16 for the proposed HMAO algorithm, 100 for GA and NSGA-II. Moreover, we have $p_e = p_s = 0.15$, $p_c = 1.0$, $p_m = 0.1$. The number of tasks is from 4 to 16 corresponding to the number of initial service nodes $4, 5, \ldots, 16$, respectively. We run each test case 10 times and compute the average results. The parameters used for the proposed HMAO algorithm, GA and NSGA-II are summarized in Table 6.

First, we simulate all test cases by the proposed HMAO algorithm, GA and NSGA-II with the specified parameters in Table 6 and obtain the average results of these three algorithms, including resource utilization, bandwidth utilization, objective value and time cost. Note that the number of initial service nodes K can influence the results of GA and NSGA-II.

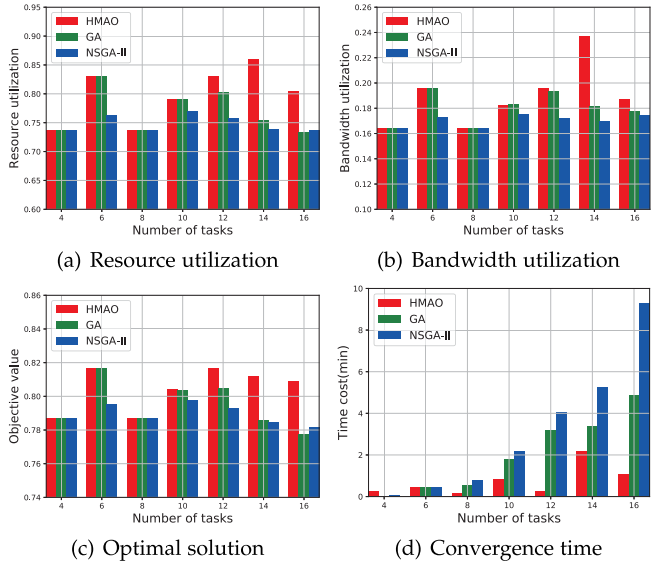
To investigate the evolution of the optimal solution for the proposed HMAO algorithm over time, we compare the evolutionary performance of the proposed HMAO algorithm, GA and NSGA-II for $L = 4, 6, 8, 10$ and the results of these three algorithms are shown in Fig. 11. It can be observed that the performance of the proposed HMAO algorithm is similar to that of GA for $L = 4, 6, 8$, NSGA-II for $L = 4, 8$. Hence, the effectiveness of the proposed HMAO algorithm is verified by comparing the performance of the proposed HMAO algorithm with GA and NSGA-II algorithms. Moreover, the convergence of the proposed HMAO algorithm is better than GA and NSGA-II as the number of tasks grows.

The reason is that the selection and exchange operators for the proposed HMAO algorithm are executed with a finer-grained sub-task level and the optimization objective tends to a good result with high probability during each iterative evolution. Besides, the probabilistic-based selection and exchange operators are used to achieve the diversity of feasible solutions and avoid early entering into the local optimal solution. However, the space of feasible solutions for GA



(a) $L = 4$ (b) $L = 6$ (c) $L = 8$ (d) $L = 10$

Fig. 11. Objective values for $L = 4, 6, 8, 10$ with HMAO, GA, and NSGA-II.



(a) Resource utilization (b) Bandwidth utilization (c) Optimal solution (d) Convergence time

Fig. 12. Performance comparisons for HMAO, GA, and NSGA-II.

and NSGA-II becomes larger with the increase in the number of tasks, i.e., it will take more time to search the optimal solution. In addition, Fig. 11 shows that GA outperforms NSGA-II for $L = 4, 6, 8, 10$, because each offspring generated by GA, which has a better fitness value than its parent, will be selected as a new individual in the next generation and that can guarantee that the performance of solving the optimization problem is improved in each iterative evolution.

In order to further analyze the performance of the proposed HMAO algorithm, we provide the maximum, minimum, mean, standard deviation and average convergence time of the simulation results for the proposed HMAO algorithm, GA and NSGA-II, and the detailed information can be observed from Table 7. In all test cases except $L = 11$, the maximum, minimum, mean and standard deviation of the optimal solutions obtained by the proposed HMAO algorithm are better than or equal to GA and NSGA-II, respectively. Furthermore, the convergence time of the proposed HMAO algorithm is slower than GA, NSGA-II for $L = 4, 6$ but faster for the other cases, the difference is more evident with increasing the number of tasks. For example, in the case of $L = 14$, we can observe that the performance of the proposed HMAO algorithm increases by 3.27 and 3.44 percent, the convergence time reduces by 35.67 and 58.38 percent for GA and NSGA-II, respectively. For $L = 16$, the mean results of the proposed HMAO algorithm improve by 4.26 and 3.49 percent, the convergence time decreases by 78.23 and 88.57 percent for GA and NSGA-II, respectively. The results can demonstrate that the proposed HMAO algorithm is an effective optimization approach for resource allocation and has a better performance in terms of solution quality, robustness and convergence.

Furthermore, we compare the results of the proposed HMAO algorithm with GA and NSGA-II in Fig. 12,

TABLE 7
Results of Simulation for HMAO, GA, and NSGA-II

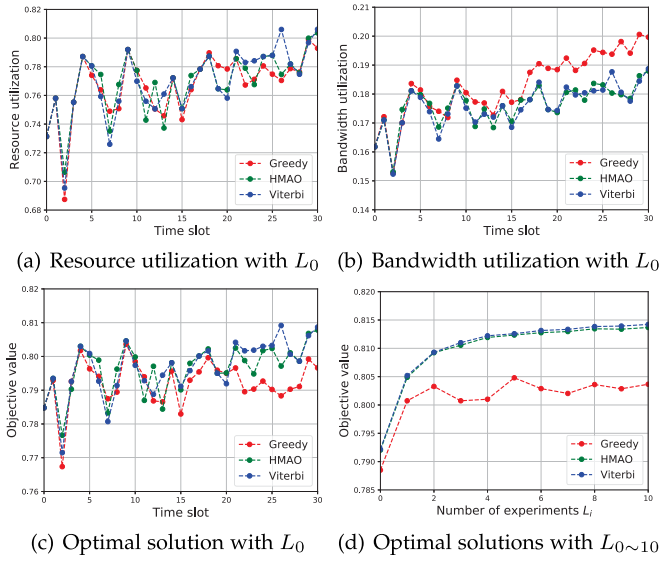
L	HMAO					GA					NSGA-II				
	Min	Max	Mean	Std	Time(min)	Min	Max	Mean	Std	Time(min)	Min	Max	Mean	Std	Time(min)
4	0.7869	0.7869	0.7869	0.0000	0.2737	0.7869	0.7869	0.7869	0.0000	0.0231	0.7869	0.7869	0.7869	0.0000	0.0569
5	0.7718	0.7718	0.7718	0.0000	0.0520	0.7718	0.7718	0.7718	0.0000	0.0481	0.7603	0.7718	0.7706	0.0034	0.1408
6	0.8170	0.8170	0.8170	0.0000	0.4752	0.817	0.817	0.817	0.0000	0.4573	0.7628	0.8170	0.7953	0.0265	0.4666
7	0.7989	0.7989	0.7989	0.0000	0.0645	0.7989	0.7989	0.7989	0.0000	0.2841	0.7567	0.7989	0.7937	0.0126	1.0380
8	0.7869	0.7869	0.7869	0.0000	0.1903	0.7869	0.7869	0.7869	0.0000	0.5710	0.7869	0.7869	0.7869	0.0000	0.8126
9	0.8170	0.8170	0.8170	0.0000	0.5219	0.8170	0.8170	0.8170	0.0000	2.3043	0.7782	0.8170	0.7899	0.0177	2.1777
10	0.8041	0.8041	0.8041	0.0000	0.8174	0.7975	0.8041	0.8034	0.0020	1.8199	0.7718	0.8041	0.7976	0.0129	2.1903
11	0.7368	0.8102	0.7874	0.0283	0.8174	0.7887	0.7944	0.7938	0.0017	1.7007	0.7771	0.7944	0.7909	0.0052	2.3302
12	0.8170	0.8170	0.8170	0.0000	0.2446	0.7818	0.8170	0.8045	0.0135	3.1743	0.7628	0.8170	0.7929	0.0161	4.0658
13	0.8070	0.8070	0.8070	0.0000	1.0464	0.7715	0.8069	0.7973	0.0104	3.8402	0.7762	0.8070	0.7977	0.0131	4.2793
14	0.8116	0.8119	0.8118	0.0001	2.1852	0.7677	0.7989	0.7861	0.0127	3.3968	0.7675	0.7989	0.7848	0.0119	5.2505
15	0.8170	0.8170	0.8170	0.0000	0.9704	0.7590	0.7923	0.7811	0.0099	4.3592	0.7680	0.8075	0.7875	0.0115	7.7129
16	0.8088	0.8088	0.8088	0.0000	1.0624	0.7576	0.7869	0.7757	0.0092	4.8812	0.7714	0.7868	0.7815	0.0049	9.3008

including resource and bandwidth utilization, optimal solution and average convergence time. Fig. 12a shows the resource utilization for different tasks, the results obtained by the proposed HMAO algorithm are equal to GA for $L = 4, 6, 8, 10$, NSGA-II for $L = 4, 8$, and better than GA and NSGA-II for the other cases. The solution with a lower resource utilization implies that there are more service nodes to be used to deploy the requested tasks. The results of bandwidth utilization are described in Fig. 12b. We can observe that the performance of the proposed HMAO algorithm is better than or equal to GA and NSGA-II for $L = 4, 6, 8, 10$. For $L = 12, 14, 16$, it is due to the fact that the number of service nodes for the proposed HMAO algorithm is less than that of the other two baseline algorithms. As the number of service nodes used by the requested tasks is smaller, the probability of placing adjacent sub-tasks of a task to different service nodes is greater, which can result in increasing the bandwidth cost. Fig. 12c illustrates the results of optimal solutions, the performance for the proposed HMAO algorithm, GA and NSGA-II is approximately equivalent with a lower number of tasks, and the proposed HMAO algorithm performs better than GA and NSGA-II as the number of tasks increases. The average convergence time of three algorithms can be observed from Fig. 12d, the time taken to converge by the proposed HMAO algorithm is worse than that of GA and NSGA-II as the number of tasks is small, but the proposed HMAO algorithm outperforms GA and NSGA-II as the number of tasks increases.

5.4 Evaluate HMAO in On-Line Resource Allocation

To further investigate the performance of the proposed HMAO algorithm in on-line resource allocation, we implement the resource allocation model in a dynamic environment and design the following experiments of dynamically allocating the available resources to the requested tasks. We set $P = 16, p_c = 1.0$ and $p_m = 0.1$ for the improved GA algorithm and $p_e = 0.15, p_s = 0.15$ and $M = 1000$ for the MAO algorithm, respectively. The total number of server nodes is a larger number $K = 64$ to guarantee that the maximum available resources are sufficient for deploying the requested tasks in each time slot, where the server nodes that are not used by the requested tasks are in sleep or shutdown states to save the operating costs and the requested tasks can be only deployed to the active server nodes. When the resource requirements of the requested tasks can not be satisfied we will wake up the server nodes from idle or shutdown states to an active state and provide the available resources for the requested tasks on demand. We assume that there are new requested tasks are appearing and old requested tasks are ending in each time slot, whose numbers are randomly produced from $L_i = \{4 * i + 1, \ldots, 4 * i + 9\}, i = 0, 1, \ldots, 10$. Each case also runs 10 times with time slots from 0 to 30 and the average results of optimal solutions are obtained by the proposed HMAO algorithm. Moreover, we compare the proposed HMAO algorithm with two heuristic algorithms of Viterbi [30] and Greedy [40] in terms of resource utilization, bandwidth utilization and optimal solution.

Viterbi. For the Viterbi algorithm in [30], the problem of placing service chains is modeled by the states and their relations as a multi-stage directed graph. We can compute the cumulative cost for each state by Viterbi algorithm. When the costs of the final stage are finished, the sequence of states, which has the minimum cost, is considered as the optimal solution.



(a) Resource utilization with L_0 (b) Bandwidth utilization with L_0

(c) Optimal solution with L_0 (d) Optimal solutions with $L_{0\sim10}$

Fig. 13. Performance comparisons for HMAO, Viterbi, and Greedy.

Greedy. The Greedy algorithm described in [40] is divided into two steps. First, the authors sort all the middleboxes in descending order with an important factor, which represents as the number of policies including the same middlebox. Second, the middleboxes are assigned to these switches iteratively. For placing a middlebox, the authors can search all available switches to calculate the cost scores accordingly and find the minimum cost to deploy the middlebox. To reduce the impact of these middleboxes that have not been placed, a weight average cost score of the unassigned middleboxes is introduced to compute the total cost score.

The simulation results for the case L_0 can be observed from Fig. 13, where the results of optimal solutions for all cases $L_0 \sim L_{10}$ are provided by the proposed HMAO, Greedy and Viterbi algorithms in Fig. 13d.

Fig. 13a shows the computing resource utilization for L_0. It can be observed that the number of computing resources are comparatively close between the proposed HMAO algorithm and the two comparison algorithms of Greedy and Viterbi in each time slot. The results of bandwidth utilization for L_0 are illustrated in Fig. 13b, where we can observe that the bandwidth utilization values obtained by the proposed HMAO algorithm are better than that of the Greedy approach, and there is average 3.63 percent performance improvement. This is because of the fact that the adjacent sub-tasks of a task can be migrated and swapped to the same service node by the proposed HMAO algorithm as far as possible in order to reduce the bandwidth resources used by the requested tasks. The proposed HMAO algorithm performs well as the Viterbi algorithm. Fig. 13c describes the objective results of optimal solution for L_0, where we can observe that the proposed HMAO algorithm performs better than the Greedy approach and the performance of the proposed HMAO algorithm shows an average increase of 0.54 percent when compared with the Greedy approach. In addition, the proposed HMAO and Viterbi algorithms

show similar results. To better analyze the performance of the proposed HMAO algorithm for processing the different number of the requested tasks in a time slot, we provide the average results of optimal solutions for $L_0 \sim L_{10}$ in Fig. 13d. It is obvious that the proposed HMAO algorithm outperforms the Greedy approach for $L_0 \sim L_{10}$, and the average performance improvement of the proposed HMAO algorithm is nearly 1.06 percent. Our proposed HMAO algorithm shows close performance to the Viterbi algorithm and the average performance difference is 0.04 percent. From Fig. 13, we can observe that the proposed HMAO algorithm is better than the Greedy approach and close to the Viterbi algorithm in dynamical resource allocation.

6 Conclusion

This paper studies the problem of resource allocation in cloud computing systems. Our aim is to maximize the resource utilization based on CPU, memory and GPU, and minimize the bandwidth cost. To address the problem, we propose the HMAO algorithm which combines the improved GA and the MAO algorithm, where the improved GA is to find an optimal resource utilization solution and the MAO algorithm is to minimize the bandwidth cost. For the MAO algorithm, we use a priority-based selection mechanism to obtain the candidate source sub-tasks, and design the selection and exchange operators by a probabilistic method to migrate and swap the sub-tasks on several service agents. The proposed HMAO algorithm can obtain the objective optimal result by cooperative co-evolutionary method.

Finally, we verify and evaluate the performance of the proposed HMAO algorithm via simulation experiments. When compared with GA and NSGA-II, we can observe that the proposed HMAO algorithm outperforms them in terms of solution quality, convergence time and robustness as the number of tasks increases. For $L = 14$, the proposed HMAO algorithm improves performance by 3.38 percent for GA and 3.44 percent for NSGA-II, and reduces the average convergence time by 19.34 percent for GA and 58.38 percent for NSGA-II. Furthermore, we compare the performance of the proposed HMAO algorithm with the Greedy and Viterbi algorithms in on-line resource allocation. We can observe that the performance of the proposed HMAO algorithm is nearly the same with the Viterbi algorithm and increases approximately by 1.06 percent for the Greedy algorithm.

References

[1] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013.

[2] S. Sezer *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.

[3] H. Erdogmus, "Cloud computing: Does Nirvana hide behind the Nebula?" *IEEE Softw.*, vol. 26, no. 2, pp. 4–6, Mar./Apr. 2009.

[4] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. 10th Int. Conf. Netw. Service Manage.*, 2014, pp. 418–423.

[5] S. M. A. Kazmi, N. H. Tran, T. M. Ho, and C. S. Hong, "Hierarchical matching game for service selection and resource purchasing in wireless network virtualization," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 121–124, Jan. 2018.

[6] H. Zheng, Y. Feng, and J. Tan, "A hybrid energy-aware resource allocation approach in cloud manufacturing environment," *IEEE Access*, vol. 5, pp. 12 648–12 656, 2017.

[7] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 2, pp. 343–356, Jun. 2017.

[8] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Trans. Commun.*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.

[9] F.-H. Tseng, X. Wang, L.-D. Chou, H.-C. Chao, and V. C. M. Leung, "Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm," *IEEE Syst. J.*, vol. 12, no. 2, pp. 1688–1699, Jun. 2018.

[10] B. Tan, H. Ma, and Y. Mei, "A NSGA-II-based approach for service resource allocation in cloud," in *Proc. IEEE Congr. Evol. Comput.*, 2017, pp. 2574–2581.

[11] S. Khebbache, M. Hadji, and D. Zeghlache, "A multi-objective non-dominated sorting genetic algorithm for VNF chains placement," in *Proc. IEEE Annu. Consum. Commun. Netw. Conf.*, 2018, pp. 1–4.

[12] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted NFV service chain deployment based on affiliation-aware vNF placement," in *Proc. IEEE Global Commun. Conf.*, 2016, pp. 1–6.

[13] E. Amaldi *et al.*, "On the computational complexity of the virtual network embedding problem," *Electron. Notes Discrete Math.*, vol. 52, pp. 213–220, 2016.

[14] B. Addis, M. Gao, and G. Carello, "On the complexity of a virtual network function placement and routing problem," *Electron. Notes Discrete Math.*, vol. 69, pp. 197–204, 2018.

[15] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[16] S. E. Dashti and A. M. Rahmani, "Dynamic VMs placement for energy efficiency by PSO in cloud computing," *J. Exp. Theor. Artif. Intell.*, vol. 28, no. 1/2, pp. 97–112, 2016.

[17] K. Mani and R. M. Krishnan, "Flexible cost based cloud resource provisioning using enhanced PSO," *Int. J. Comput. Intell. Res.*, vol. 13, no. 6, pp. 1441–1453, 2017.

[18] P.-Y. Yin and J.-Y. Wang, "Ant colony optimization for the nonlinear resource allocation problem," *Appl. Math. Comput.*, vol. 174, no. 2, pp. 1438–1453, 2006.

[19] B. Muthulakshmi and K. Somasundaram, "A hybrid ABC-SA based optimized scheduling and resource allocation for cloud environment," *Cluster Comput.*, vol. 22, pp. 10769–10777, 2019.

[20] M. N. Omidvar, X. Li, and K. Tang, "Designing benchmark problems for large-scale continuous optimization," *Inf. Sci.*, vol. 316, pp. 419–436, 2015.

[21] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up fast evolutionary programming with cooperative coevolution," in *Proc. IEEE Congr. Evol. Comput.*, 2001, pp. 1101–1108.

[22] X. Wu, Y. Wang, J. Liu, and N. Fan, "A new hybrid algorithm for solving large scale global optimization problems," *IEEE Access*, vol. 7, pp. 103 354–103 364, 2019.

[23] Z. Ren, Y. Liang, A. Zhang, Y. Yang, Z. Feng, and L. Wang, "Boosting cooperative coevolution for large scale optimization with a fine-grained computation resource allocation strategy," *IEEE Trans. Cybern.*, vol. 49, no. 12, pp. 4180–4193, Dec. 2019.

[24] M. Yang *et al.*, "Efficient resource allocation in cooperative co-evolution for large-scale global optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 4, pp. 493–505, Aug. 2017.

[25] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Auton. Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[26] T. Kaihara, "Multi-agent based supply chain modelling with dynamic environment," *Int. J. Prod. Econ.*, vol. 85, no. 2, pp. 263–269, 2003.

[27] M. M. de Weerdt, Y. Zhang, and T. Klos, "Multiagent task allocation in social networks," *Auton. Agents Multi-Agent Syst.*, vol. 25, no. 1, pp. 46–86, 2012.

[28] W. Wang, Y. Jiang, and W. Wu, "Multiagent-based resource allocation for energy minimization in cloud computing systems," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 47, no. 2, pp. 205–220, Feb. 2017.

[29] X.-L. Zheng and L. Wang, "A multi-agent optimization algorithm for resource constrained project scheduling problem," *Expert Syst. Appl.*, vol. 42, no. 15/16, pp. 6039–6049, 2015.

[30] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[31] B. Kar, E. H.-K. Wu, and Y.-D. Lin, "Energy cost optimization in dynamic placement of virtualized network function chains," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 372–386, Mar. 2018.
[32] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evol. Comput.*, vol. 2, no. 3, pp. 221–248, Sep. 1994.
[33] L. F. Nawaf, S. M. Allen, and O. Rana, "Optimizing infrastructure placement in wireless mesh networks using NSGA-II," in *Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun. IEEE 16th Int. Conf. Smart City IEEE 4th Int. Conf. Data Sci. Syst.*, 2018, pp. 1669–1676.
[34] N. M. Razali *et al.*, "Genetic algorithm performance with different selection strategies in solving TSP," in *Proc. World Congr. Eng.*, 2011, pp. 1–6.
[35] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Nav. Res. Logistics*, vol. 45, no. 7, pp. 733–750, 1998.
[36] L. Nawaf, S. M. Allen, and O. Rana, "Internet transit access point placement and bandwidth allocation in wireless mesh networks," in *Proc. IEEE 7th Annu. Comput. Commun. Workshop Conf.*, 2017, pp. 1–8.
[37] D. Bhamare *et al.*, "Optimal virtual network function placement in multi-cloud service function chaining architecture," *Comput. Commun.*, vol. 102, pp. 1–16, 2017.
[38] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, "Modeling and placement of cloud services with internal structure," *IEEE Trans. Cloud Comput.*, vol. 4, no. 4, pp. 429–439, Oct.–Dec. 2016.
[39] S. Kaur and A. Verma, "An efficient approach to genetic algorithm for task scheduling in cloud computing environment," *Int. J. Inf. Technol. Comput. Sci.*, vol. 4, no. 10, pp. 74–79, 2012.
[40] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin, "Improve service chaining performance with optimized middlebox placement," *IEEE Trans. Services Comput.*, vol. 10, no. 4, pp. 560–573, Jul./Aug. 2017.

Xiangqiang Gao received the BSc degree from the School of Electronic Engineering, Xidian University, Xi'an, China, in 2012, and the MSc degree from Xi'an Microelectrinics Technology Institute, Xi' an, China, in 2015. He is currently working toward the PhD degree with the School of Electronic and Information Engineering, Beihang University, Beijing, China. His research interests include rateless codes, software defined network, and network function virtualization.

Rongke Liu (Senior Member, IEEE) received the BS and PhD degrees from Beihang University, Beijing, China, in 1996 and 2002, respectively. He was a visiting professor with the Florida Institution of Technology, USA, in 2006; The University of Tokyo, Japan, in 2015; and the University of Edinburgh, U.K., in 2018, respectively. He is currently a full professor with the School of Electronic and Information Engineering, Beihang University. He received the support of the New Century Excellent Talents Program from the Minister of Education, China. He has attended many special programs, such as China Terrestrial Digital Broadcast Standard. He has authored or co-authored more than 100 papers in international conferences and journals. He has been granted more than 20 patents. His research interest include wireless communication and space information network.

Aryan Kaushik (Member, IEEE) received the MSc degree in telecommunications from the Hong Kong University of Science and Technology, Hong Kong, in 2015, and the PhD degree in communications engineering from the Institute for Digital Communications, School of Engineering, University of Edinburgh, Edinburgh, U.K, in 2020. He is currently a research fellow in communications and radar transmission with the Institute of Communications and Connected Systems, University College London, U.K. He has held visiting research appointments with the Wireless Communications and Signal Processing Lab, Imperial College London, U.K, from 2019–20, the Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg, in 2018, and the School of Electronic and Information Engineering, Beihang University, China, from 2017–19. His research interests include broadly in signal processing, radar, wireless communications, millimeter wave, and multi-antenna communications.