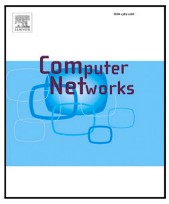
Computer Networks 257 (2025) 110932

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Cooperation-based server deployment strategy in mobile edge computing system

Xin Li [a,*], Meiyan Teng [a], Yanling Bu [a], Jianjun Qiu [a], Xiaolin Qin [a], Jie Wu [b]

[a] CCST, Nanjing University of Aeronautics and Astronautics, Nanjing, 211106, China
[b] Center for Networked Computing, Temple University, Philadelphia, PA 19122, USA

ARTICLE INFO

Keywords:
Cooperation
Edge server deployment
Mobile edge computing
Utility optimization

ABSTRACT

In our exploration of Mobile Edge Computing (MEC) systems, we address the critical challenge of edge server deployment, aiming to enhance application responsiveness through optimized server placement and cooperation. Our study diverges from traditional approaches that prioritize server location, instead highlighting the untapped potential of server collaboration for sharing computing resources. This cooperative strategy not only boosts resource utilization and trims response times but also intricately complicates deployment strategies. We introduce an innovative Collaboration-Based Server Deployment (CBSD) algorithm that stands out by facilitating cooperative communication between edge servers via Base Stations (BSs), even under stringent resource constraints. This algorithm employs a dual-phase approach: initially utilizing a non-collaborative *Gradient* algorithm for resource allocation among cooperative regions, followed by a strategic distribution of resources based on regional demand. Our comprehensive simulations show that our proposed methodology improves system utility and throughput by 35% and 25%, respectively, while robustness reaches 90% compared to the baseline. These results represent improvement in managing limited edge resources effectively.

1. Introduction

With the rapid development of mobile network technology and the Internet of Things (IoT), more and more low-latency application scenarios need to be considered. Cloud Computing (CC) has begun to shift to Mobile Edge Computing (MEC) [1] that sinks the computing power to the distributed edge servers closer to the user to improve the real-time performance [2], reduce the energy consumption [3], enhance security privacy protection, and pursue the quality of services (QoS) [4]. It expresses that small-scale and decentralized server deployment could improve the system utility [5]. However, deploying numerous edge servers everywhere is infeasible due to constraints on resources and costs. Therefore, factors such as the deployment location, number, and computing power of servers significantly impact the system utility and, by extension, the QoS [6]. Consequently, it is essential to explore server deployment strategies to improve MEC system performance, particularly focusing on optimizing the number of servers and the distribution of their resources.

In the MEC system depicted in Fig. 1, edge servers are typically located at base stations (BSs), each covering a distinct region. Inter-base station communication interfaces, as discussed in [7], facilitate communication and computing collaboration between servers [8,9]. This architecture enables servers to forward requests to other regions, exemplifying the collaborative relationship between servers. However, load distribution varies across different regions [10,11], for example, residential and office regions experience distinct loads. Additionally, load distribution is time-dependent, with residential areas witnessing different loads during day and night. Consequently, the number of requests in each region is both different and dynamic, reflecting the changing load distribution caused by user mobility [12]. A larger load, indicative of more requests in a region, requires a successful response that depends on the substantial computing resources of the server [10]. Therefore, to execute all requests and enhance the QoS, we investigate the cooperation-based server deployment problem to determine the computing resources allocated to each server in every region.

However, the cooperation-based server deployment problem faces multiple challenges in the MEC system. First, we must know the connectivity and network transmission quality between servers, and the server collaboration enables migrating services to other connectable servers for execution, which increases the complexity of solving the server deployment problem. Another, forward requests will increase the response time, so it must limit the number of hops for service request cooperation because each hop will incur additional response delay. Next, the movement of users causes the load of servers to be unevenly distributed, which leads to the server deployment strategy

* Corresponding author.
E-mail address: lics@nuaa.edu.cn (X. Li).

https://doi.org/10.1016/j.comnet.2024.110932
Received 29 July 2024; Received in revised form 6 November 2024; Accepted 16 November 2024
Available online 24 November 2024

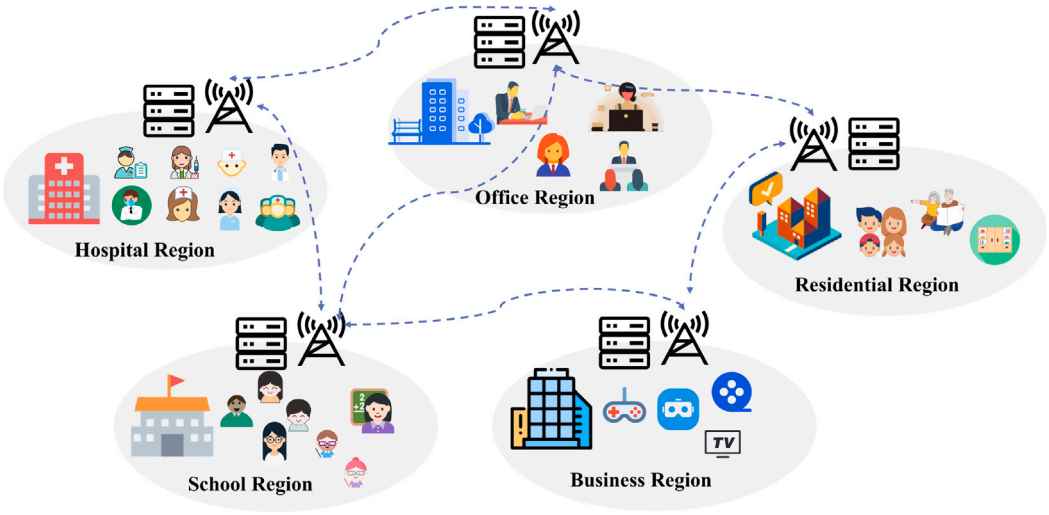


Fig. 1. The Mobile Edge Computing System.

being dynamically adjusted to adapt to the varied load. Finally, the delay tolerance and the resource demand of tasks requested by users are diverse and the deployment cost is limited, which also increases the difficulty of making server deployment decisions. Nowadays, a lot of leading research about edge computing focuses on computation offloading [13,14], service placement [15], service migration [16,17], Edge Intelligence (EI) [18,19], etc. These studies focus on the application level, whereas server deployment addresses server configuration issues. Consequently, these research efforts assume that edge servers are already in place. A sound server deployment strategy not only ensures the optimal operating environment for services but also underscores the fundamental value of our study to the field.

Regarding the server deployment problem, most works are aiming to minimize the cost [20,21], the network resource [22,23], task time and energy consumption [24], and achieve load balance [25], etc [26]. Apart from those, we defined a system utility related to resource utilization and delay time, where the resource refers to a resource block consisting of network resources and computing resources. The server deployment in work [20] mainly determines the number and location of services, while other works [21,26] only determine the location of servers. However, these works ignore the amount of resources allocated to servers is an important factor that affects the quality of service. Other works [22,27,28] consider resource utilization but are limited to network resources, ignoring that computing resources are also an important factor affecting server deployment. Different from the above works, the server deployment we study focuses on solving computing resource deployment strategy under server collaboration conditions. Additionally, existing studies [25,28] solve server deployment problems based on load, and the work [28] increases dynamically the network resources, but they forget the feature that the regional load periodically changes.

In this paper, we first establish a server collaboration mechanism that considers for the dynamic changes in regional load and server locations, facilitating the assessment of server collaboration feasibility. Subsequently, we examine the periodic variation of load and leverage server collaboration to achieve complementary load change patterns and resource sharing among different servers, which improves the resource utilization of servers. Therefore, we proposed a Cooperation-based Edge Server Deployment Strategy (CBSD) to maximize the system utility under the limited cost constraint. In summary, the main contributions can be summarized as follows:

(1) We give a detailed description of the features of regional load periodic variation in a mobile edge computing system and exploit the load complementarity of different regions to discover server collaboration technology. Then, we quantify some of the system parameters, define an optimization objective for measuring the effectiveness of the deployment strategy and formulate the server deployment problem. Finally, we define a system utility based on the task types including delay-sensitive and delay-tolerant tasks.
(2) In order to avoid task execution failures or long delays caused by server-collaborative computing, we propose a server-collaboration judgment mechanism. On one hand, we base it on the system characteristics, task attributes, and cost constraints to set the cooperative conditions of servers. On the other hand, we stipulate a collaborative response mode for tasks to further measure the system utility.
(3) To solve the collaboration server deployment problems, we use the flower tree algorithm to bind the collaborative servers, which is converted to a non-collaborative server deployment problem and solved by the *Gradient* algorithm we proposed. For the resource allocation problem among collaborative servers, we present a collaboration-based server deployment (CBSD) algorithm to allocate computing resources based on load proportion.
(4) We perform extensive simulation experiments to verify the performance of the CBSD algorithm in terms of system utility, throughput, and robustness under different system settings. The results show that our algorithms have excellent performance in improving the system utility, can successfully accept a larger number of tasks in terms of throughput, and have a higher robustness.

The remainder of this article is organized as follows. We overview the related works on server deployment in Section 2, and introduce the MEC system and formulate the server deployment problem in Section 3. Section 4 presents the cooperation way of servers, including the cooperative conditions of servers and a collaborative response mode for tasks. Section 5 proposes the CBSD algorithm and the *Gradient* algorithm to solve the collaboration server deployment problems. In Section 6, we validate our algorithms by analyzing the experimental results. Finally, Section 7 summarizes all works of our paper.

2. Related works

Edge computing can essentially solve the problems in the traditional cloud computing process [29], but different edge server deployment schemes will have a more significant impact on system utility. To obtain an optimal system utility, many scholars have conducted in-depth research on server deployment using mathematical models and optimization algorithms to optimize solutions from multiple aspects.

Since server providers require a lot of cost to deploy servers, most researchers studied server deployment problem with the goal of minimizing the cost. The authors [20] used queuing theory and vector quantization(VQ) technique to proposed an optimal edge servers deployment strategy to optimize the number and the location of servers and the users allocation, which minimizes the cost of services provider and guarantees the completion time of all services requests. Similarly, the literature [21] also aims at minimizing costs, which uses stochastic games to solve multi-user computation offloading and edge server deployment respectively. Beside, there are also some studies [22,23] on the deployment of edge servers from the perspective of network resource optimization to improve the utilization of network resources. These algorithms utilized the relaxation constraints idea to solve the optimal server deployment strategy and used the genetic search algorithm to determine the optimal deployment location of nodes. Finally, they took advantage of the Lagrangian operator to solve the best service node set.

Apart from the above objectives, the authors in [24] defined a task experience function as an evaluation index from two aspects: task time and energy overhead. And the literature [25] minimized the distances between servers, while taking into account capacity constraints for load balancing and enabling workload sharing between servers. Finally, the literature [26] formulated the six-objective server deployment optimization model and proposed a many-objective evolutionary algorithm to optimize the transmission delay, workload balancing, energy consumption, deployment costs, network reliability, and server quantity.

However, the existing technical solutions have a shortcoming, where they do not take into account the cycle difference of the regional load. They ignore the load variation pattern among different regions may be complementary, leading to the increase in the number of edge servers deployed. In our previous work [30], we have discussed the server deployment based on load gradient changes. However, it had no consideration for server cooperation. In this paper, we take the server cooperation into account since the BSs can communicate through interfaces. We define the cooperation conditions based on the delay requirements of the service and the network conditions. In scenarios that do not meet the requirements of collaboration, we analyze the load variation rule in the edge area to obtain the relationship between the utility gradient and the load variation, and propose a gradient-aware server deployment strategy. But in the collaboration scenario, we take advantage of greedy thinking to propose a collaboration-based server deployment strategy.

3. System model and problem formulation

3.1. The system model

Fig. 2 shows a mobile edge computing (MEC) system with edge server collaboration, where the edge servers can be deployed near the BSs and they can connect to each other through interfaces in the Radio Access Network (RAN). Under normal circumstances, the requests of User Equipment (UE) can only be responded to by the local edge server, such as request 1 only be responded to by local server 1 in Fig. 2. However, server 1 and server 2 can communicate with each other, so server 1 can forward the request 1 to server 2 to respond, which achieves collaboration between server 1 and server 2. Similarly, server 2 and service 3 can also cooperate to respond to request 2. Therefore, the servers can communicate with each other to realize data transmission, and then support server collaborative computing and service migration.

The MEC system can adopt diversified deployment methods based on actual conditions, which produces different utility and brings challenges to the study of edge servers deployment. The requests can be directly executed on the local edge server to reduce delay and meet the real-time requirements of delay-sensitive tasks. At the same time, the requests can be transmitted to the other edge servers. Server collaborative computing will generate additional time overhead, which will affect the task response quality. In addition, the limited computing power of edge servers is an important factor affecting the task response, and further influences the system utility. In order to evaluate the influence of edge server computing power allocation on the system, we defined utility variables. Next, we will give descriptions of regions, tasks, and utility functions based on system parameters (see Table 1).

Table 1
Summary of key symbol.

Symbol	Description
$\mathbf{M}$	The region set
m	A region m, $m \in \mathbf{M}$
$\mathbf{L}$	The load set
l_m	The load of region m, $l_m \in \mathbf{L}$
λ	The computing power matrix
λ_m	The computing power allocated to region m
N	A cycle is divided into N stages
$\mathbf{J}$	The task set
Y_j	The characteristics of task j
j	A task j, $j \in \mathbf{J}$
ξ_j	The identification of the task j
μ_j	The utility from the execution of task j
m_j	The region where the task j belongs to
τ_j	The completion time deadline of task j
θ_j	The actual execution time of task j
k_j	The category of task j
$\eta_j(\theta_j)$	The utility function of task j
α	The coefficient of delay-sensitive tasks
ω	The coefficient of delay-tolerant tasks
Ω	The total utility in a cycle
Φ	The total computing power

3.2. System parameters

Region Description: In order to reflect the law of load changes, this paper divides the entire MEC system into multiple regions. An region may contain multiple BSs, and the load changes in these regions are different. However, the load variation in a short period of time may not be obvious. The purpose of the period division is also to reflect the periodic changes of the load. The characteristics of regions are described by a tuple Λ, as follows:

$$\Lambda =< \mathbf{M}, \mathbf{L}, \lambda > \tag{1}$$

where, $\mathbf{M}$ is the region set and M is the number of regions, $m \in \mathbf{M}$ is the unique identifier of the region m, and $l_m \in \mathbf{L}$ represents the load in the current region m. The λ is the matrix of computing power allocated to each regions, where λ_m is the computing power allocated to region m. In order to characterize the cyclical variation of the load, a day is regarded as a cycle, which matches the real life scenarios. A cycle is divided into N stages. For example, a day can be divided into 4 stages, and every 6 h constitutes a stage.

Tasks Description: We divide all tasks into two categories based on real-time requirements: delay-sensitive tasks and delay-tolerant tasks. For delay-sensitive tasks, if the response time does not meet their real-time requirements, it will reduce the quality of the task. So the delay-sensitive tasks need to be executed in edge servers preferentially. But for delay-tolerant tasks, a longer delay has little effect on their quality. In many cases, it is also possible to place them in the cloud data center. This means that delay-sensitive tasks have a higher execution priority than delay-tolerant tasks, especially when the edge server load is high. For example, when the load of the edge server reaches a preset threshold, the edge server must refuse to perform delay-tolerant tasks and reserve computing resources for delay-sensitive tasks to ensure their quality of service. In this paper, the tasks set is denoted by $\mathbf{J}$, and the characteristics of a task $j \in \mathbf{J}$ are expressed as:

$$Y_j =< \xi_j, \mu_j, m_j, \tau_j, \theta_j, k_j > \tag{2}$$

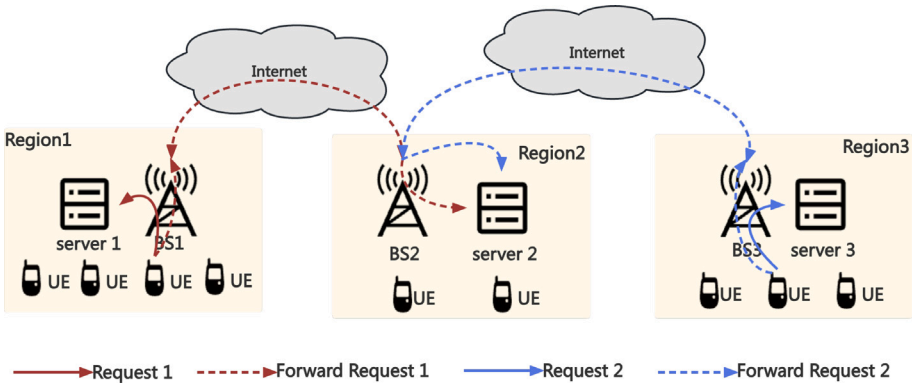
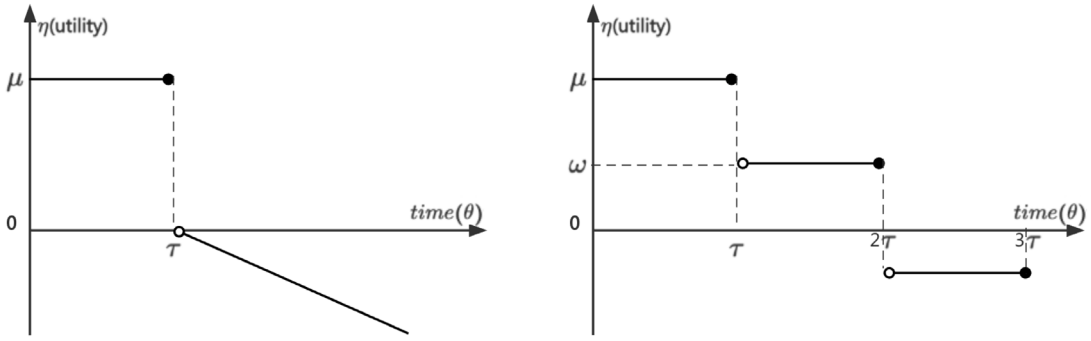


Fig. 2. An edge server collaboration system.



(a) the delay-sensitive tasks (b) the delay-tolerant tasks

Fig. 3. The utility function of different kinds of tasks.

where, the ξ_j is used as the identification of the task j, and μ_j represents the utility from the successful execution of the task j. The region where the task j belongs to is marked as $m_j \in \mathbf{M}$, the completion time deadline of the task j is τ_j, and the actual execution time of the task j is θ_j. The symbol $k_j \in \{0, 1\}$ is the category of task j, where $k_j = 1$ means the j is a delay-sensitive task and $k_j = 0$ represents the j is a delay-tolerant task.

The Utility Function: The objective of our research is to maximize the system utility under the premise of ensuring the quality of all tasks through reasonable edge server deployment. Because the completion time required by different tasks is different, and the actual execution time of all tasks is also different. In addition, the actual execution time of different types of tasks also has a different impact on the system utility. So we construct a utility function, as shown in Fig. 3, which describes the effect of the completion time of delay-sensitive tasks and delay-tolerant tasks on the system benefit. From Fig. 3, we find that tasks with different types have different completion utilities and the same tasks with different completion times also get different utilities. In this paper, we suppose that when the server load reaches the preset threshold, the delay-tolerant task must be rejected to ensure the delay requirement of the delay-sensitive task. That is, when the load of the edge server reaches a certain percentage, the delay-sensitive tasks can preferentially occupy edge computing resources.

For delay-sensitive tasks that respond before the completion time of the task deadline, the system can get the full utility. However, if the execution time exceeds the deadline, which means that the real-time requirements cannot be met, the utility will directly become negative. And the longer the time exceeded, the worse the service quality becomes. The equation of utility function for the delay-sensitive tasks is defined in Eq. (3) as shown in Fig. 3(a), where τ_j is the completion time required by the task j, θ_j is the actual execution time of task j, and α is the attenuation coefficient of the utility function when the deadline is exceeded.

$$\eta_j(\theta_j) = \begin{cases} \mu_j, & \theta_j \le \tau_j \\ -\alpha(\theta_j - \tau_j), & otherwise \end{cases} \tag{3}$$

For delay-tolerant tasks, they usually have periodic characteristics. When the required completion time is exceeded, the utility decreases periodically, as shown in Fig. 3(b). Therefore, the utility function of the delay-tolerant task can be expressed as Eq. (4), which is a piecewise function. And the ω is the attenuation coefficient of utility for the delay-tolerant tasks, which is ω is less than the $\frac{1}{2}\mu_j$. So, when the actual execution time $\theta_j \in (2\tau_j, 3\tau_j]$, the utility will be negative.

$$\eta_j(\theta_j) = \mu_j - (\mu_j - \omega)\left\lfloor \frac{\theta_j}{\tau_j} \right\rfloor \tag{4}$$

3.3. The problem formulation and analysis

The purpose of our research is to obtain a reasonable server deployment strategy to maximize the execution utility of the system, under limited computing resources. From Eqs. (3)–(4), the utility of a task is related to the execution time of this task, which is determined by the computational power of the server, i.e., $\theta_j \to \lambda$. The utility function $\eta_j(\theta_j)$ of task j is rewritten as $\eta_j(\lambda)$. For a stage, the total utility is the accumulation of the utility of all tasks is defined as

$$h_n(\lambda) = \sum_{m=1}^{M} \sum_{j=1}^{J} \eta_j(\lambda) \tag{5}$$

Our research aims to maximize the utility of a cycle, where a cycle is divided into N stages, as shown in Eq. (6). Assume that the total computing power of the system initially owned is Φ, and the computing

power allocated to the M regions in turn is $\lambda = \lambda_1, \lambda_2, \ldots, \lambda_M$. The sum of computing power allocated to each region cannot exceed the total computing power. So the object and constraint of server deployment problem can be expressed as:

$$\max \Omega = \sum_{n=1}^{N} h_n(\lambda) \tag{6}$$

$$s.t. \quad \lambda_1 + \lambda_2 + \cdots + \lambda_M \leq \Phi \tag{7}$$

Through the extension and analysis of the above formulas, we show and prove that the resource allocation problem of the edge server is NP-hard.

Theorem 1. *In the MEC system, the resource allocation problem of edge servers is NP-hard.*

Proof. The MEC system is as described above. Under the constraint that the total computing resource is Φ, appropriate computing resources are allocated to M sub-regions, which can be decomposed into a knapsack problem.

For the knapsack problem, a set of items $\cup B_i (1 \leq i \leq k)$ is given, where the weight w_i corresponds to the value v_i. The next step is to select items in the knapsack to maximize the total value under the premise that the total weight does not exceed Π. In the scenario of this article, we first construct a set of regions $\cup \Lambda_k (1 \leq k \leq M)$, $\Pi = \Phi$, and its value changes with the computing resources allocated to each region, which can be expressed as $\eta(Y_j)$. At the same time, the goal of the research is to maximize the overall utility under different deployment strategies. Finally, let $\eta(Y_j)$ be a constant, and the problem presented in this paper is completely equivalent to the ordinary knapsack problem. The knapsack problem is NP-hard, so the resource allocation problem of edge servers in the MEC system is also NP-hard.

4. The server cooperative mode

In the MEC system, the servers can communicate with each other through interfaces of BSs, which achieve the server cooperation to response task. However, servers must be able to collaborate for task computing under certain conditions, which are influenced by many factors. What is more, the task collaborative response mode is an important factor in the server deployment strategy, and further influence the system utility. In order to get a reasonable server deployment strategy, this section first discusses the necessary conditions for server cooperation and the task collaborative response mode, laying a foundation for the following deployment strategy design.

4.1. The conditions for server cooperation

In this section, we analyze the server cooperation conditions from three aspects: physical link, cooperation overhead, and complementary load.

(1) Physical Link Communication Condition. Because edge server collaboration to response tasks requires data transmission between servers, there must exist a physical link to guarantee server communication. In a MEC system, BSs can communicate with each other through the physical interface. The BSs are connected by physical links to ensure data transmission and improve the efficiency of cooperation. Therefore, one of the necessary conditions for collaboration between edge servers is that there is an interface link between the BSs where the servers are deployed nearby. As shown in Fig. 2, server 1 and server 2, server 2 and server 3 have been connected by physical links, which meet the conditions of cooperation. However, there is no physical link communication between server 1 and server 3, so they cannot cooperate.

(2) Collaboration Delay Condition. If there is a physical link between the servers, the task will be responded to collaboratively through data transmission or service migration, however, this will bring additional communication costs and response delays. For delay-sensitive tasks, the additional delay may reduce utility. Therefore, another cooperative communication condition is that the total response time from submission to final execution time for a task must be less than its computing deadline time. We define the collaboration delay condition using Eq. (8), where set S denote the distance between the base stations, B is the data transmission rate and the $\frac{S}{B}$ is the data transmission time. The ϵ is the task transfer time, so, the $2 * (\frac{S}{B} + \epsilon)$ represents additional communication time for task collaboration computing. The actual computing time of tasks is θ and the deadline completion time of tasks is τ. The formulation shows that the distance S between BSs is the most important factor affecting communication time, so the collaborative delay condition also means that the server distance should not be too long. Otherwise, it will cause too long response time, which will not meet the real-time requirements of delay-sensitive tasks and further affect their service quality.

$$2 * (\frac{S}{B} + \epsilon) + \theta \leq \tau \tag{8}$$

(3) Complementary Load Condition. A cycle is divided into N stages, and the load of each stage of one region is different. At the same stage, the server with a larger load cannot satisfy the deadline response time of all tasks, which will reduce the system utility. However, the servers with little load will have resources left. In order to increase the system utility, the server with a larger load can transfer a part of the load to the server with little load to collaborate computing. Due to the data transmission and service migration that can cause additional time, we set a necessary condition for collaborative computing between two servers, which is that the two regions have at least two phases with complementary loads in a cycle to cooperate. For example, we suppose that a day is one cycle and it is divided into 6 stages, which means each 4 h is a stage. In Fig. 2, the load of region 1 during the 8 h–12 h period is 100, and the load of region 2 is 20. During the 12 h–16 h, the load of region 1 is 30 and the load of region 2 is 80. In this case, the region 1 and the region 2 are complementary to satisfy the two-stage load in one cycle. The load of region 1 can be transferred to region 2 that has less load during 8 h–12 h to avoid wasting the server resource of region 2.

4.2. The server cooperation algorithm

In this paper, we assume that each region can only collaborate with one region at most, and we transform the server collaboration problem into a graph matching problem. The servers in all regions are seen as the nodes in the graph. If the regions can cooperate, the nodes in the graph also have corresponding edge connections. In order to improve the efficiency of collaboration, we hope that the number of collaborative groups is the largest, that is, the number of graph connections is the largest. So the server deployment problem is converted to the maximum matching number of the graph, and the matching conditions of the graph are the three necessary conditions given above. For the typical maximum matching problem of bipartite graphs, the Hungarian algorithm can be used [31]. However, this algorithm is not suitable for the scenario shown in this paper, as the nodes of the region do not have the bipartite nature and thus the graph is not equivalent to the bipartite graph. Actually, the problem can be classified as the maximum matching problem of general graphs and we thus solve it using the blossom algorithm [32].

We combine the above three necessary conditions and adopt the flowered tree algorithm to get the server collaboration algorithm, which is shown in Algorithm 1. Among them, line 3 determines whether the two regions are physically connected, and line 6 determines whether the distance S between the base stations meets the cooperation delay condition. The third load complementary condition is in line 6 of this algorithm, which determines whether there are

Algorithm 1: Server Cooperation Algorithm (SCA)

Input: Information for all regions:
M: the number of regions in system; $\cup\delta^i = \{\delta_1^i, \delta_2^i, \ldots, \delta_N^i\}$:the average load set of N stages in the region i;
Λ_i: Location coordinates of the region i is (x_i, y_i);
$1 \le i \le M$.
Output: List of regions where servers can collaborate.

```
1  for each 1 ≤ i ≤ M do
2      for each 1 ≤ j ≤ M do
3          if i ≠ j and Λ_i isConnect Λ_j then
4              S ← sqrt((x_i − x_j)^2 + (y_i − y_j)^2);
5              W ← (τ − θ − 2 ∗ ε) ∗ B/2;
6              if S ≤ W then
7                  if complementary(δ_n^i, δ_n^j) then
8                      loc ← getloc(Λ_i, Λ_j);
9                      bloosom(Λ_i, Λ_j, loc);
10                     match[i] ← j;
11                 else
12                     break;
13                 end
14             else
15                 break;
16             end
17         else
18             break;
19         end
20     end
21 end
```

at least two phases of load complementary in one cycle of the two regions. Lines 8–10 utilize the flowering tree algorithm to calculate the maximum matching number of the general graph and return the result to ensure maximum collaboration and improve overall deployment efficiency.

4.3. The cooperative response mode for tasks

From Eqs. (3)–(5), we can find that the system utility is related to the actual response time of the task. Therefore, we need to set a cooperative response mode for tasks to calculate their actual response time. The corresponding pattern of delay-sensitive tasks is different from that of delay-tolerant tasks. At the same time, we combine the remaining resources of servers and the server cooperation list obtained by Algorithm 1, and then give different task collaboration response modes according to the different delay characteristics of tasks. The cooperative response mode for tasks includes:

(1) The system determines whether the local edge server of the task has remaining computing resources. If so, the task will be executed on the local server. Otherwise, perform (2).
(2) The system judges whether the server has a collaboration region according to Algorithm 1. If it does not exist, go to step (3). Otherwise, based on the task type and the load situation of the collaborative region, the response mode on the collaborative server is divided into the following three situations:

- The load of the collaboration server is less than ε_1, which shows that the load of the collaboration server is very little. Therefore, no matter what type of task, they can be responded to on the collaborative server.
- The load of the collaboration server is higher than ε_1 but less than ε_2, which shows that the load of the collaboration server is higher. At this time, the collaborative server prioritizes select delay-sensitive tasks to execution, but rejects delay-tolerant tasks.
- The load of the collaboration server is higher than ε_2, which means that the load of the collaboration server is very high. In order to ensure the quality of service for delay-sensitive tasks in its own region, the collaboration server will reject all requests that will be transferred to the cloud (go to step (3)).

(3) The task is scheduled to the cloud center for execution.

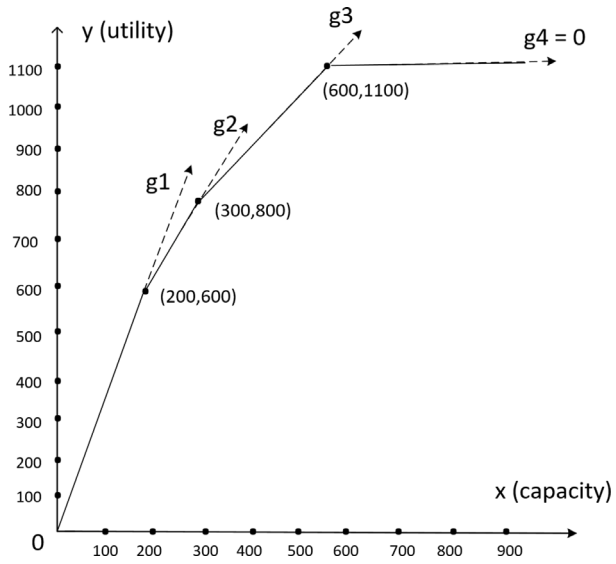


Fig. 4. A simplified utility function.

5. Edge server deployment strategy in MEC system

Based on the system parameters and problem definitions in chapter 3, the conditions for server cooperation in Section 4.1, we propose the server cooperation algorithm to adjust whether the servers can cooperate in part 4.2, and give the cooperative response mode for tasks in part 4.3. Next, we first introduce a non-cooperative server deployment algorithm, named the *Gradient* algorithm. And then, we propose a cooperation-based edge server deployment algorithm in the MEC system.

5.1. The non-cooperative server deployment algorithm

For scenarios where servers cannot collaborate, we adopt the *Gradient* algorithm proposed in our previous work [30]. For a region $k \in \mathbf{M}$, we assume that this region is allocated with λ_k computing power, and the loads in the N stages of this region are described as, $\delta_1, \delta_2, \ldots, \delta_N$, respectively. The system utility is related to the allocated computing power and load, so we set the utility in a stage of a region as the minimum of the computing power and the load. Furthermore, our goal is to maximize the total utility of the system, that is, the sum utilities in N stages of M regions, as shown in Eq. (9).

$$\max U = \sum_{k=1}^{M} h(\Lambda_k) = \sum_{k=1}^{M}\sum_{i=1}^{N} \min(\delta_i, \lambda_k) \tag{9}$$

In order to understand the utility function more clearly, we use a simple example for illustration. We simplify the scenario and divide one cycle of a region into 3 stages ($N = 3$). Assume that the average load of the three stages of the region in a cycle is 200, 300, and 600,

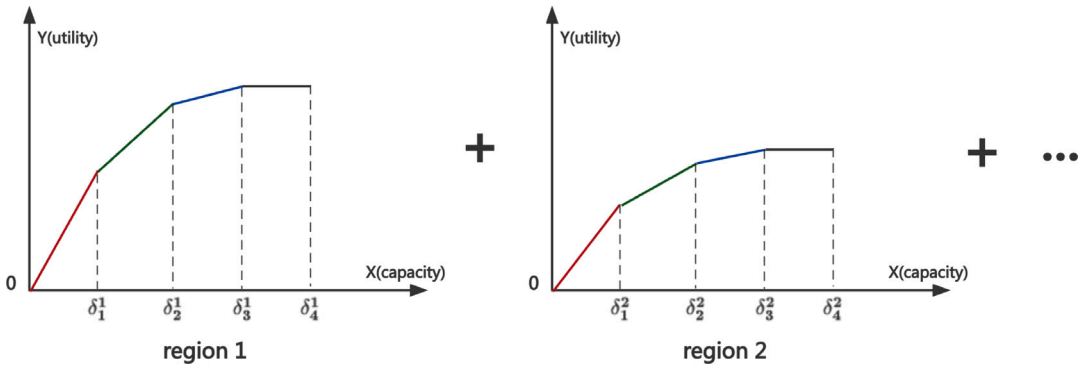


Fig. 5. The total utility of the system.

Algorithm 2: Gradient Algorithm

Input: Information for all regions and load:
M: the number of regions in system; $\cup\delta^i = \{\delta^i_1, \delta^i_2, \ldots, \delta^i_N\}$:the average load set of N stages in the region $i, 1 \le i \le M$;
Ω: the total computing resources; $usedCapicity$: the allocated computing resources, the initial setting is 0.
Output: $\cup\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_M\}$: the computing resources allocated by M regions.

```
1  for each 1 ≤ n ≤ N do
2      for each 1 ≤ i ≤ M do
3          if usedCapicity < Ω then
4              if λ_i < δ^i_n then
5                  usedCapicity ← usedCapicity + (δ^i_n − λ_i);
6                  λ_i ← δ^i_n;
7              else
8                  continue;
9              end
10             i++;
11         else
12             break;
13         end
14     end
15     n++;
16 end
```

and the computing resources allocated in a region are x. Therefore, the total utility of this region is:

$$h(\Lambda_k) = min(200, x) + min(300, x) + min(600, x) \tag{10}$$

The expression of the utility function is shown in Fig. 4, which shows that the utility function of a region is a segment function and the value of the utility changes with the allocated computing resources. This segment function is divided into four parts, which correspond to the four different stages of load in one cycle of this region. As the allocated computing resources increase, the value of the function gradually increases. But when the allocated computing resources reach a certain number, the total utility remains stable, which means that it is useless to allocate more computing resources, and the utility reaches the maximum value at this time. From Fig. 4, the gradient of each stage can be obtained, assuming that they are respectively g_1, g_2, g_3, and g_4, and the values of these four gradients are sequentially reduced. When the allocated computing resources are greater than the maximum load of N stages in a cycle, the gradient is 0, and no more resources should be allocated.

In this paper, the entire system contains M regions, and the total utility of the entire system is the sum of the utility of all regions. Therefore, the utility function of the entire system is shown in Fig. 5. We use different colors to indicate the utility growth rate at each stage. It can be seen from Fig. 5 that the $[0, 200]$ part is the fastest growing part, and the $[200, 400]$, the $[400, 600]$, and the $[600, 800]$ parts grow more and more slowly. Therefore, the *Gradient* algorithm prioritizes allocating the computing resources to the $[0, 200]$ part, followed by the other parts, which ensures that the system get more utilities.

Based on the above analysis, the main idea of the *Gradient* algorithm proposed in this paper is based on the feature that the growth rate of the utility function will be slower as the load increases, giving priority to the region with little load. Because the region with little load is more effective for the rapid growth of utility. The *Gradient* algorithm, detailed in Algorithm 2, systematically allocates computing resources across regions, selecting the optimal region iteratively until resources are depleted, as outlined in lines 1–3. Analysis of Eqs. (9)–(10) indicates that if allocated resources fall below the current load, allocation continues; otherwise, no further resources are allocated, as detailed in lines 4–9. Specifically, if the allocated resources are insufficient, allocation continues based on the load distribution, and the used resources are updated accordingly, as detailed in lines 5–6.

5.2. Cooperation-based server deployment (CBSD) algorithm

The core idea of the cooperation-based server deployment (CBSD) algorithm is task migration through collaborative servers to share load, thereby reducing the number of edge servers deployed and improving the utilization of computing resources. The CBSD algorithm merges the two collaborative regions calculated by Algorithm 1 into one region, which is a non-collaborative scenario, and uses the *Gradient* algorithm as shown in Algorithm 2 to obtain the computing resources allocated to the merged region. Then, the CBSD algorithm will continue to decide the ratio of computing resources allocated to the collaborative regions based on the load distribution, whose detailed process as shown in Algorithm 3.

Step 1 Calculate the cooperation information (line 2) of the region using the server cooperation algorithm (Algorithm 1).

Step 2 Bind the collaboration region (lines 3–10). If the region Λ_i and Λ_j satisfy the cooperation conditions, the cooperation region marks Λ_{ij} as shown in line 6, whose load is the sum of the load of Λ_i and Λ_j, defined by δ^{ij}_n in line 7.

Step 3 Allocate computing resources (lines 11–17). The cooperation region obtained in Step 2 is regarded as an independent region, whose resource allocation λ_{ij} is calculated by *Gradient* algorithm given in the following Section 5.1.

Step 4 Re-allocate the collaborative region (lines 19–24). First, the load ratio of sub-region Λ_i is defined by $\frac{\delta^i_n}{\delta^{ij}_n}$, where δ^i_n is the load of Λ_i and the δ^{ij}_n is the total load of collaborative region Λ_{ij}. Then, allocate the total computing resources λ_{ij} obtained in Step 3 to Λ_i based on load ratio.

Algorithm 3: Cooperation-based Server Deployment (CBSD) Algorithm

Input: Information for all regions and load:
M: the number of regions in system; $\cup\delta^i = \{\delta_1^i, \delta_2^i, \ldots, \delta_N^i\}$:the average load set of N stages in the region $i, 1 \le i \le M$;
Ω: the total computing resources; $usedCapicity$: the allocated computing resources, the initial setting is 0.
Output: $\cup\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_M\}$: the computing resources allocated by M regions.

```
for n ≤ N do
    ServerCooperationAlgorithm();
    for each Λ_i ∈ ∪Λ do
        for each Λ_j ∈ ∪Λ do
            if Λ_i cooperation with Λ_j then
                Λ_ij ← Λ_i + Λ_j;
                δ_n^ij ← δ_n^i + δ_n^j;
            end
        end
    end
    for each Λ_ij do
        if usedCapicity < Ω then
            λ_ij = GradientAlgorithm();
        else
            break;
        end
    end
    for each Λ_i ∈ ∪Λ do
        if Λ_i cooperation with Λ_j then
            λ_i ← (δ_n^i / δ_n^ij) λ_ij;
            λ_j ← (δ_n^j / δ_n^ij) λ_ij;
        end
    end
end
```

5.3. The complexity of the CBSD algorithm

In this section, we analyze the complexity of the CBSD Algorithm, which includes four steps. The step 1 calculates the server cooperation, whose complexity is $O(M^2)$. Next, in the step 2, the CBSD will bind the collaboration region, it also has $O(M^2)$ complexity. Then step 3 uses the *Gradient* algorithm to calculate the computing resources allocated to the binding regions. In the optimal case, all regions cooperate with each other, and the complexity of *Gradient* algorithm at a stage is $O(M/2)$. However, in the worst case, all regions cannot meet the cooperation conditions with each other, the complexity of *Gradient* algorithm at a stage is $O(M)$. For the step 4, it achieves the computing resources allocated to each regions, whose complexity is $O(M)$. Therefore, for a stage, the complexity of CBSD algorithm is $O(2M^2+2M)$ in the optimal case, and the complexity in the worse case is $O(2M^2+3/2M)$. Finally, because a cycle is divided into N stages, the complexity of CBSD algorithm is $O(NM(2M+2))$ in the optimal case, and the complexity in the worse case is $O(NM(2M+3/2))$.

6. Evaluation

The experiment randomly generated several different tasks, each with different completion time deadline, utilities, etc., and took the number of requests in each region as the load, and the load of all regions will change over time. The experimental parameter settings are detailed in Table 2, which includes the utility loss coefficient $\alpha = 0.3$ and the number of sub-regions M, set at values of 5, 10, and 15, respectively. The number of stages in a cycle is $N = 4$, which means that a cycle of 24 h is divided into 4 stages, and 6 h is a stage. The load threshold for edge servers ϵ is set at 0.8, while the thresholds for collaborative servers are initialized at $\epsilon_1 = 0.9$ and $\epsilon_1 = 0.5$, respectively. The initial total computing resources is $\Phi = 100$ and increase by 100 with each iteration.

Table 2
Parameter settings.

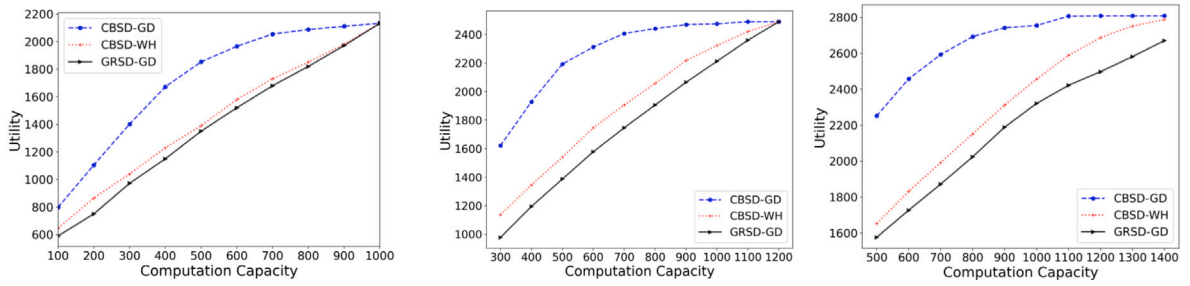
Symbol	Description	Value
α	The utility loss coefficient	0.3
M	The number of regions	5, 10, 15
N	The number of stages	4
ϵ	The non-collaboration load threshold	0.8
ϵ_1	The collaboration upper threshold	0.9
ϵ	The collaboration lower threshold	0.5
Φ	The initial total computing resources	100

Since the server deployment problem we studied is based on fixed total computing resources and a set number of requests, the overall system energy consumption is predetermined [33,34]. We should focus on enhancing task response quality under the established conditions of resources and load. So, we evaluate the algorithm performance from three aspects: system utility, throughput, and robustness. In order to evaluate the effectiveness of the proposed algorithm, we carry out a large number of simulation experiments and compare the CBSD-GD we proposed with the CBSD-WH and GRSD-GD algorithms. The specific descriptions are as follows:

- CBSD-GD: The algorithm proposed in this paper combines the *Gradient* algorithm (as detailed in Algorithm 2) and the cooperation-based server deployment (CBSD) algorithm (as detailed in Algorithm 3), which is used as the evaluation object.
- CBSD-WH: This algorithm represents a combination of the CBSD algorithm proposed in this article and the traditional *Weight* algorithm. The *Weight* algorithm calculates the average workload per cycle for each sub-region, sums these to establish the region's total workload, and then allocates computing capacity proportionally based on each sub-region's contribution to the total. The experimental settings of the *Weight* algorithm can refer to the paper [30].
- GRSD-GD: The algorithm uses GRSD algorithm and *Gradient* algorithm in server collaboration scenarios. Among them, the GRSD algorithm adopts a greedy strategy when performing service collaboration. The greedy strategy collaborates whenever the server has extra computing power, regardless of the task type.

6.1. The deployment utility analysis

In this paper, a large number of experiments are carried out, we first evaluate the utility of our proposed CBSD-GD algorithm by comparing it with CBSD-WH and GRSD-GD algorithms. The detailed definition of system utility is provided in Eqs. (3)–(6), located in Section 3.2. In Fig. 6, we set different numbers of regions and analyze the changes of system utility with the variation of allocated computing resources. Fig. 6(a) compares the utility among the three algorithms when the number of regions is $M = 5$ in the system, where the x-axis represents the total available computing resources in the system and the y-axis represents the total utility of each algorithm under the current setting. It can be seen from Fig. 6(a) that the utility of the CBSD-GD algorithm is always greater than or equal to the other deployment algorithms at various number of computing resources, which indicates that the CBSD-GD algorithm has a good performance in the deployment of edge servers. It is worth noting that when the allocated computing resource are greater than 1000, the utility of the three deployment algorithms



(a) the utility comparison $M = 5$ (b) the utility comparison $M = 10$ (c) the utility comparison $M = 15$

Fig. 6. The utility comparison.

tends to be the same. The main reason is the response time deadline of all tasks can be satisfied when the allocated computing resources are enough, and the system utility will become constant. Fig. 6(b) shows the changing of system utility of the three algorithms when the number of regions is $M = 10$. The general trend of the three algorithms is that as the allocated computing resources increase, the utility will gradually increase, and eventually the three will achieve the same utility. But the CBSD-GD algorithm is always larger than the other two algorithms. Fig. 6(c) shows the system utility of the three algorithms when the number of regions is $M = 15$, where the changing trend of utility is the same as in the previous two cases. In addition, the utility of the algorithm CBSD-GD is always higher than that of GRSD-GD, which shows that prioritizing delay-sensitive tasks can improve the overall quality of services during collaboration. Fig. 6 indicates that under resource constraints, our algorithm improves utility by up to 35% compared to other algorithms.

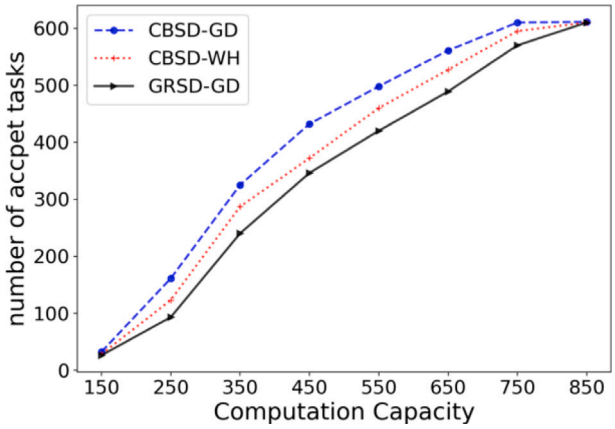


Fig. 7. The edge-side throughput.

6.2. The edge-side throughput analysis

Traditional deployment algorithms mostly target task execution delay, energy consumption, etc., and do not care about the utilization of edge server computing resources. In this section, we execute many experiments to verify the throughput performance of various algorithms on the edge side. Throughput, defined as the total number of tasks per time unit [35], is calculated using the formula $(\sum_{n=1}^{N}\sum_{m=1}^{M}\delta_n^m)/T$, where T denotes the total time unit across N stages. In the MEC system, greater throughput on the edge side means that the edge servers have more powerful computing processing capabilities and higher computing resource utilization. Because the computing resources of traditional edge servers are relatively limited, it is very important for the MEC system to improve the computing throughput on the edge side. Fig. 7 shows the total number of tasks successfully accepted by the three deployment algorithms under the different computing resources. It can be seen from Fig. 7 that the number of tasks accepted by the three algorithms is almost the same when the system has few allocated resources, which is because most tasks cannot be met in this situation. With the increase of allocated computing resource, the CBSD-GD algorithm has demonstrated its superiority. The number of tasks received by our algorithm at the edge is always greater than or equal to the other algorithms, which shows that the CBSD-GD we proposed has good edge throughput performance. For computing power within the range of 250 to 750, our algorithm improves throughput by approximately 10%–25%. As computing resources continue to increase, the number of tasks accepted by the three algorithms is eventually the same. This is because too many computing resources are allocated, and the computing power on the edge servers becomes so large that all tasks are received.

6.3. The robustness analysis of the algorithm

In addition to analyzing the reliability of the algorithm, this section analyzes the robustness of the CBSD-GD algorithm we proposed, which is defined that when the average load changes within a certain range, the load change will not affect the performance of the deployment algorithm. That is to say, in the case of load changes, the algorithm performance has good stability. Fig. 8 illustrates the robustness of the three algorithms, with the x-axis representing system settings, including the number of regions and allocated computing resources. The y-axis shows the utility ratio, defined as the actual utility compared to the maximum possible utility under specific settings. It can be seen from Fig. 6 and Fig. 7 that the CBSD-GD algorithm can obtain the best utility compared to the other algorithms, at the same time, the performance also has good stability seen from Fig. 8. It can be found from Fig. 8 that when the computing resource is 700, no matter whether the number of regions is 5, 10, or 15, the actual utility of the CBSD-GD algorithm we proposed is about 90% of the maximum utility that the system can harvest in this setting. And if the computing resource is 800, the actual utility of the algorithm is about 95% of the maximum utility. However, the utility ratios for the CBSD-WH and GRSD-GD algorithms range from approximately 70% to 80%. The performance of the proposed CBSD-GD algorithm shows an improvement of about 10% to 20% over these comparison algorithms. As the number of regions and the load changes, the utility of the algorithm proposed in this paper are relatively stable, which shows that the CBSD-GD algorithm has good robustness.

As shown in Figs. 6–8, under conditions of extremely insufficient or extremely sufficient resources, the performance of the CBSD-GD algorithm is comparable to that of the other algorithms. Therefore, the algorithm proposed in this paper achieves optimal performance under non-extreme resource conditions and is well-suited for realistic scenarios with uneven load distribution and collaborative servers.

7. Conclusions

This article identifies the patterns of load variation and server collaboration characteristics, using complementary load collaborations for

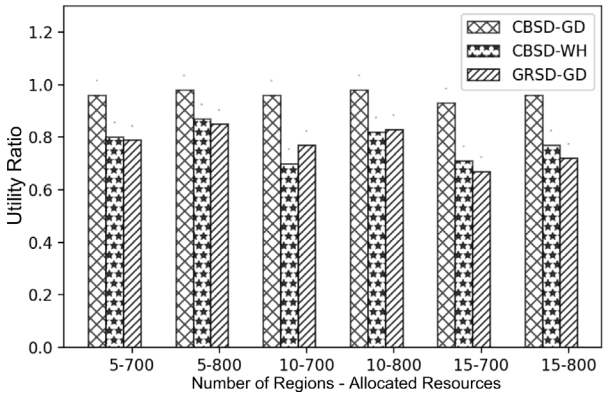


Fig. 8. The robustness comparison.

service migration and load sharing to enhance resource utilization in edge servers. First, we set the conditions for server collaboration from three aspects: physical link, collaboration delay, and complementary load. Then, we proposed the server cooperation algorithm and the cooperative response mode for tasks based on the kind of tasks. Next, we analyzed the rule of load variation and proposed a CBSD algorithm that utilized the *Gradient* algorithm to allocate computing resources. Finally, simulation experiments verified that the CBSD-GD algorithm we proposed had good performance in three aspects: system efficiency, throughput, and robustness. In the future work, we aim to implement the algorithm in real-world scenarios, gather load data from base stations to improve experiments, and use machine learning to dynamically adjust resource allocation, facilitating adaptive and intelligent server deployment.

CRediT authorship contribution statement

Xin Li: Writing – review & editing, Supervision, Project administration, Conceptualization. **Meiyan Teng:** Writing – original draft. **Yanling Bu:** Writing – review & editing. **Jianjun Qiu:** Methodology. **Xiaolin Qin:** Writing – review & editing. **Jie Wu:** Writing – review & editing.

Funding

This work was supported in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] Y. Siriwardhana, P. Porambage, M. Liyanage, M. Ylianttila, A survey on mobile augmented reality with 5G mobile edge computing: Architectures, applications, and technical aspects, IEEE Commun. Surv. Tutor. 23 (2) (2021) 1160–1192.

[2] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu, L. Guo, Computation offloading in mobile edge computing networks: A survey, J. Netw. Comput. Appl. 202 (2022) 103366.

[3] P. Cong, J. Zhou, L. Li, K. Cao, T. Wei, K. Li, A survey of hierarchical energy optimization for mobile edge computing: A perspective from end devices to the cloud, ACM Comput. Surv. 53 (2) (2020) 1–44.

[4] Y. Yin, Z. Cao, Y. Xu, H. Gao, R. Li, Z. Mai, QoS prediction for service recommendation with features learning in mobile edge computing environment, IEEE Trans. Cogn. Commun. Netw. 6 (4) (2020) 1136–1145.

[5] Y. Chen, D. Wang, N. Wu, Z. Xiang, Mobility-aware edge server placement for mobile edge computing, Comput. Commun. (2023).

[6] J. Li, W. Liang, W. Xu, Z. Xu, X. Jia, W. Zhou, J. Zhao, Maximizing user service satisfaction for delay-sensitive IoT applications in edge computing, IEEE Trans. Parallel Distrib. Syst. 33 (5) (2022) 1199–1212.

[7] W. Li, J. Chen, Y. Li, Z. Wen, J. Peng, X. Wu, Mobile edge server deployment towards task offloading in mobile edge computing: A clustering approach, Mob. Netw. Appl. 27 (4) (2022) 1476–1489.

[8] H. Ye, B. Cao, J. Liu, P. Li, B. Tang, Z. Peng, An edge server deployment method based on optimal benefit and genetic algorithm, J. Cloud Comput. 12 (1) (2023) 148.

[9] M. Su, G. Wang, K.-K.R. Choo, et al., Prediction-based resource deployment and task scheduling in edge-cloud collaborative computing, Wirel. Commun. Mob. Comput. 2022 (2022).

[10] X. Xu, Q. Huang, X. Yin, M. Abbasi, M.R. Khosravi, L. Qi, Intelligent offloading for collaborative smart city services in edge computing, IEEE Internet Things J. 7 (9) (2020) 7919–7927.

[11] W.-Z. Zhang, I.A. Elgendy, M. Hammad, A.M. Iliyasu, X. Du, M. Guizani, A.A. Abd El-Latif, Secure and optimized load balancing for multitier IoT and edge-cloud computing systems, IEEE Internet Things J. 8 (10) (2020) 8119–8132.

[12] Z. Ning, P. Dong, X. Wang, S. Wang, X. Hu, S. Guo, T. Qiu, B. Hu, R.Y. Kwok, Distributed and dynamic service placement in pervasive edge computing networks, IEEE Trans. Parallel Distrib. Syst. 32 (6) (2020) 1277–1292.

[13] A. Feng, H. Ge, Y. Wang, J. Wang, W. Li, K. Liu, Mobile edge computing offloading strategy based on improved BP neural network, in: 2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics, ICCCBDA, IEEE, 2020, pp. 138–143.

[14] S. Zhou, W. Jadoon, The partial computation offloading strategy based on game theory for multi-user in mobile edge computing environment, Comput. Netw. 178 (2020) 107334.

[15] M. Teng, X. Li, X. Qin, J. Wu, Priority based service placement strategy in heterogeneous mobile edge computing, in: International Conference on Algorithms and Architectures for Parallel Processing, Springer, 2020, pp. 314–329.

[16] R.A. Addad, D.L.C. Dutra, M. Bagaa, T. Taleb, H. Flinck, Fast service migration in 5G trends and scenarios, IEEE Netw. 34 (2) (2020) 92–98.

[17] I. Labriji, F. Meneghello, D. Cecchinato, S. Sesia, E. Perraud, E.C. Strinati, M. Rossi, Mobility aware and dynamic migration of MEC services for the internet of vehicles, IEEE Trans. Netw. Serv. Manag. 18 (1) (2021) 570–584.

[18] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, A.Y. Zomaya, Edge intelligence: The confluence of edge computing and artificial intelligence, IEEE Internet Things J. 7 (8) (2020) 7457–7469.

[19] Y. Liu, M. Peng, G. Shou, Y. Chen, S. Chen, Toward edge intelligence: Multiaccess edge computing for 5G and Internet of Things, IEEE Internet Things J. 7 (8) (2020) 6722–6747.

[20] B. Li, P. Hou, H. Wu, F. Hou, Optimal edge server deployment and allocation strategy in 5G ultra-dense networking environments, Pervasive Mob. Comput. 72 (2021) 101312.

[21] Z. Ning, Y. Yang, X. Wang, L. Guo, X. Gao, S. Guo, G. Wang, Dynamic computation offloading and server deployment for UAV-enabled multi-access edge computing, IEEE Trans. Mob. Comput. (2021).

[22] H. Li, Y. Liu, Z. Yang, F. Meng, D. Wang, Y. Nan, Maximizing network resource utilization based server deployment algorithm in LTE-edge computing, in: 2020 IEEE 3rd International Conference on Computer and Communication Engineering Technology, CCET, IEEE, 2020, pp. 244–248.

[23] W. Du, H. Sun, H. Wang, X. Guo, B. Zou, Server deployment algorithm for maximizing utilization of network resources under fog computing, in: International Conference on Computer Engineering and Networks, Springer, 2020, pp. 1466–1474.

[24] B. Li, P. Hou, K. Wang, Z. Peng, S. Jin, L. Niu, Deployment of edge servers in 5G cellular networks, Trans. Emerg. Telecommun. Technol. 33 (8) (2022) e3937.

[25] T. Lähderanta, T. Leppänen, L. Ruha, L. Lovén, E. Harjula, M. Ylianttila, J. Riekki, M.J. Sillanpää, Edge computing server placement with capacitated location allocation, J. Parallel Distrib. Comput. 153 (2021) 130–149.

[26] B. Cao, S. Fan, J. Zhao, S. Tian, Z. Zheng, Y. Yan, P. Yang, Large-scale many-objective deployment optimization of edge servers, IEEE Trans. Intell. Transp. Syst. 22 (6) (2021) 3841–3849.

[27] T.K. Rodrigues, K. Suto, N. Kato, Edge cloud server deployment with transmission power control through machine learning for 6G internet of things, IEEE Trans. Emerg. Top. Comput. 9 (4) (2021) 2099–2108, http://dx.doi.org/10.1109/TETC.2019.2963091.

[28] J. Liu, H. Xu, G. Zhao, C. Qian, X. Fan, L. Huang, Incremental server deployment for scalable NFV-enabled networks, in: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 2020, pp. 2361–2370, http://dx.doi.org/10.1109/INFOCOM41043.2020.9155364.

[29] M.M. Sadeeq, N.M. Abdulkareem, S.R. Zeebaree, D.M. Ahmed, A.S. Sami, R.R. Zebari, IoT and Cloud computing issues, challenges and opportunities: A review, Qubahan Acad. J. 1 (2) (2021) 1–7.

[30] J. Qiu, X. Li, X. Qin, H. Wang, Y. Cheng, Utility-aware edge server deployment in mobile edge computing, in: Algorithms and Architectures for Parallel Processing: 19th International Conference, ICA3PP 2019, Melbourne, VIC, Australia, December 9–11, 2019, Proceedings, Part I 19, Springer, 2020, pp. 359–372.

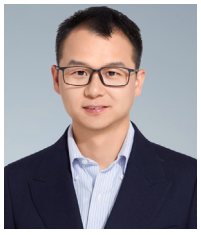
[31] A.Q. Abduljaleel, et al., Reviewer assignment using weighted matching and hungarian algorithm, Turk. J. Comput. Math. Educ. (TURCOMAT) 12 (7) (2021) 619–627.

[32] A.F. Ocampo, M.-R. Fida, J.F. Botero, A. Elmokashfi, H. Bryhni, Opportunistic CPU sharing in mobile edge computing deploying the cloud-RAN, IEEE Trans. Netw. Serv. Manag. (2023).

[33] H. Rezaie, M. Golsorkhtabaramiri, A shared channel access protocol with energy saving in hybrid radio-frequency identification networks and wireless sensor networks for use in the internet of things platform, IET Radar Sonar Navig. 17 (11) (2023) 1654–1663.

[34] M. Teng, X. Li, K. Zhu, Joint optimization of sequential task offloading and service deployment in end-edge-cloud system for energy efficiency, IEEE Trans. Sustain. Comput. 9 (3) (2024) 283–298.

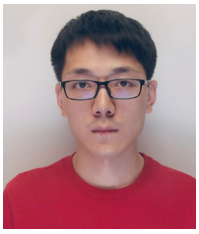
[35] H. Rezaie, M. Golsorkhtabaramiri, A fair reader collision avoidance protocol for RFID dense reader environments, Wirel. Netw. 24 (6) (2018) 1953–1964.



Xin Li received the B.S. and Ph.D degrees from Nanjing University in 2008 and 2014, respectively. Currently, he is an associate professor in the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. His research interests include cloud computing, edge computing, and distributed computing.

Meiyan Teng received the B.S. degree from Nanjing University of Information Science & Technology in 2018. She received the M.S. degree from Nanjing University of Aeronautics and Astronautics (NUAA). At present, she is a Ph.D. student at the College of Computer Science and Technology, NUAA. Her research interests include edge computing and edge intelligence.

Yanling Bu received her Ph.D. degree in computer science from Nanjing University, China in 2022. She is currently an associate researcher with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. Her research interests include Internet of Things (IoT), mobile sensing, and RFID systems.



Jianjun Qiu received the B.S degree in Anhui University of Agriculture (AHAU) in 2018 and M.S degree from Nanjing University of Aeronautics and Astronautics (NUAA) in 2021. He used to be a software engineer at NetEase and is now a software engineer at ByteDance. He is interested in Cloud Native, Could Computing and Edge Computing.

Xiaolin Qin is a professor in the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics. His research interests include cloud data management and knowledge discovery.

Jie Wu is the director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the director of International Affairs at College of Science and Technology. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. He is an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a fellow of the AAAS and a fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.