Collaborative Task Processing for Multiple Edge Nodes Based on Service Placement

Lei Shi
Shilong Feng (fsl@mail.hfut.edu.cn)
Rui Ji
Juan Xu
Xu Ding

Research Article

Keywords: Mobile Edge Computing, Collaborative Task Processing, Service Placement, Resource Allocation, Lyapunov Optimization.

Posted Date: January 1st, 2024

DOI: https://doi.org/10.21203/rs.3.rs-3804016/v1

Additional Declarations: No competing interests reported.

Collaborative Task Processing for Multiple Edge Nodes Based on Service Placement

Lei Shi[1,2], Shilong Feng[1,2*], Rui Ji[1,2], Juan Xu[1,2], Xu Ding[1,3]

[1*]School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, 230009, Anhui, China.

[2]Engineering Research Center of Safety Critical Industrial Measurement and Control Technology, Ministry of Education, Hefei, 230009, Anhui, China.

[3]Institute of Industry & Equipment Technology, Hefei University of Technology, Hefei, 230009, Anhui, China.

*Corresponding author(s). E-mail(s): fsl@mail.hfut.edu.cn;

Abstract

With the rapid development of the Internet of Things (IoT), Mobile Edge Computing (MEC) technology is shifting computational and storage capabilities from centralized clouds to the network edge to meet the low-latency demands of numerous emerging applications. However, ensuring quality of service (QoS) for mobile users becomes challenging in dense, decentralized wireless communication environments and with limited MEC server storage capacity. Against this background, this paper proposes a collaborative task processing model for multiple ENs based on service placement and formulates a MINLP optimization problem aimed at minimizing system latency and cost. To address this problem, we introduce an online optimization algorithm (OPDA) based on the Lyapunov framework which operates in real-time without the need to predict future information. Subsequently, we decompose the long-term optimization problem into a series of one-time slot problems and design a two-stage one-time slot optimization algorithm to obtain an approximate optimal solution. Specifically, we use the Lagrange multiplier approach to solve the resource allocation problem for tasks and the matching theory to solve the offloading decision and service placement problem for tasks. Simulation results show that our algorithm can achieve near-optimal latency performance while satisfying long-term cost constraints.

Keywords: Mobile Edge Computing, Collaborative Task Processing, Service Placement, Resource Allocation, Lyapunov Optimization.

1 Introduction

As the Internet of Things (IoT) continues to thrive, there is a growing trend of deploying resource-intensive tasks on user devices, including mobile games, video analytics, and virtual/augmented reality (VR/AR). However, the limited computational and storage capacities of these devices often struggle to meet the processing requirements of such user tasks [1]. In response to this challenge, cloud computing technology has emerged as an effective solution, enabling task offloading from local devices to remote cloud processing. Yet, cloud computing is not without its drawbacks, such as the extensive data transmission between users and remote clouds, and the considerable round-trip distances involved. These factors contribute to an extended system delay for task processing, subsequently escalating offloading costs [2]. Hence, cloud computing technology may not be the optimal solution for processing latency-sensitive tasks.

In response to this issue, the European Telecommunications Standards Institute (ETSI) has introduced a novel computing paradigm - Mobile Edge Computing (MEC) technology [3]. This approach extends cloud computing to the edge of the network and has garnered significant attention within the academic community [4]. By offloading tasks from user devices to proximate edge nodes, MEC provides users with services that are low-latency and high-bandwidth, thereby overcoming the challenges traditional cloud computing faces when dealing with large-scale, real-time data [5]-[6]. In essence, MEC technology offers more efficient, real-time, and secure data processing solutions across various industries, and is set to play a pivotal role in the future development of the IoT. However, MEC servers, in comparison to remote cloud servers, have limited communication and computing capabilities. This makes it difficult to meet the high volume of offloading demands from users during peak load periods, leading to a less than optimal user experience. As a result, the collaboration between cloud computing and edge computing has emerged as a new research trend in order to more effectively handle tasks [7]. For tasks that are sensitive to delay, offloading them to the edge for processing is a viable solution, while tasks that can tolerate delay can be offloaded to the remote cloud for processing [8]. Simultaneously, optimizing task offloading decisions and resource allocation to minimize task processing delay and cost has become a critical aspect of cloud-edge collaborative processing for computing tasks.

Furthermore, edge caching technology, a rapidly evolving field, is garnering increasing interest [9]. The fundamental concept of this technology involves caching a variety of services that users may require on servers during off-peak times. These services can either be downloaded from the remote cloud when user tasks arrive or pre-cached on MEC servers [10]. Unlike traditional edge caching, the services cached in this scenario refer to those deployed on MEC servers, including relevant databases (such as datasets required for user computing tasks), to execute tasks offloaded from users. Only MEC servers equipped with the corresponding services can perform tasks uploaded by users, thereby reducing the delay and energy consumption associated with user tasks [11].

While most literature on edge caching theoretically presumes that MEC servers can cache all types of services, practical limitations such as storage capacity mean that MEC servers must prioritize user-required services within their capacity constraints[12]. Furthermore, research into the collaborative caching of services across multiple MEC servers is not yet comprehensive. In our previous work, we did not consider the issue of services required by users placement, that is, services may be randomly assigned to any MEC server. Therefore, service storage may be uneven across multiple MEC servers, potentially causing some servers to be overloaded while others are underutilized. To address task offloading issues in vehicle edge computing networks, a load balancing scheme was proposed [13], which leverages collaboration between multiple MEC servers and cloud servers to jointly process tasks, thereby reducing task processing delay. Consequently, under the constraints of limited storage resources, determining which services should be cached on which server to maintain system balance is of paramount importance [14]-[15].

In order to solve the load imbalance problem between MEC servers and fulfill the latency requirements of different user tasks, we propose a multi-MEC server collaborative task processing model based on service placement. This issue is divided into two sub-problems. The first is the task offloading decision problem, which involves which MEC server should handle the tasks that users offload to the edge. The second is the service placement problem, which involves which MEC server should store the services required by users. The goal of this paper is to minimize system latency and cost by making optimal offloading decisions for tasks and allocating reasonable computational resources, as well as optimizing service placement decisions. The primary contributions of this paper are as follows:

1) Under the constraint of satisfying the long-term average system cost, we aim to minimize system latency by making optimal offloading decisions for tasks and allocating reasonable computing resources, while optimizing service placement decisions.
2) In response to the aforementioned issue, this paper proposes an online optimization algorithm (OPDA) based on the Lyapunov framework. This algorithm operates in real-time and does not require the prediction of future information.
3) The paper utilizes the Lagrangian multiplier method to solve the computational resource allocation problem for tasks and employs matching theory to address the task offloading decisions and service placement issues.
4) Simulation results indicate that the proposed approach not only achieves near-optimal latency performance but also maintains lower system costs.

The organization of the remaining sections in this paper is as follows: Section 2 summarizes relevant research work. Section 3 introduces the system model and formulates the optimization problem addressed in this paper. Section 4 outlines the design of the OPDA online algorithm to address the proposed optimization problem. Section 5 conducts an in-depth analysis of the performance of the OPDA algorithm through simulation results. Finally, Section 6 provides a comprehensive summary of the entire paper.

2 Related Work

2.1 Computing Offloading and Resource Allocation in MEC

In recent years, MEC has garnered considerable attention, with numerous researchers making significant contributions in this field. A key theme in both mobile cloud computing and mobile edge computing revolves around how to offload latency-sensitive tasks while adhering to energy consumption constraints [16]-[18]. Existing literature primarily focuses on task offloading decisions. For example, in [19], authors examined the issue of energy-efficient dynamic offloading in MEC for the IoT, with the goal of minimizing average transmission energy consumption while ensuring device performance. In [20], authors delved into the optimization of minimizing bandwidth consumption during task offloading in IoT applications, striking a balance to maximize the reliability of these applications. In [21], authors investigated the task offloading problem in ultra-dense networks and proposed a heuristic greedy offloading scheme aimed at minimizing the energy consumption of mobile devices within acceptable time constraints. The aforementioned studies primarily concentrate on optimizing task offloading decisions, with less focus on the judicious allocation of communication and computational resources.

Several other studies have focused on offloading user tasks to MEC servers for processing, without considering the collaborative processing of user-uploaded tasks between cloud and edge computing. For instance, in [22], the authors proposed a delay-optimized resource allocation method with the aim of minimizing communication latency between mobile devices and MEC servers. They then provided performance metrics for tasks from three perspectives: average delay, levels of energy consumption, and the availability of computational resources. In [23], a joint task offloading and resource allocation scheme in MEC, assisted by moving vehicles, was proposed. This method allocates tasks to edge servers at base stations or vehicles, while efficiently distributing the resources of base stations to minimize the overall weighted task processing delay for all devices. In [25], the authors explored the computation offloading problem in MEC networks with dynamically weighted tasks. They proposed a Computation Offloading algorithm based on Deep Supervised Learning to jointly optimize the offloading decisions and bandwidth allocation, thereby minimizing the system utility of the MEC network.

While other studies have considered the scenario of collaborative task processing between a single MEC server and a remote cloud server, they often overlook the collaboration among multiple MEC servers. For instance, in [26], the authors proposed a cloud-edge collaborative computing framework with the aim of minimizing task processing delay by leveraging both remote cloud and edge resources. In [27], the authors took into account the cooperation scenarios of cloud computing and MEC within networks with limited communication capabilities. They designed an iterative heuristic MEC resource allocation algorithm to address the problem of multi-user computing offloading.

2.2 Service Caching in MEC

Recently, numerous studies have suggested the use of edge caching technology within the MEC system to minimize task processing delay or energy consumption [28]-[30]. In [28], the authors introduced an energy-efficient task caching and offloading scheme that strikes a balance between the resource utilization of MEC servers and user experience. By storing either part or all of the task data on the MEC server, task execution efficiency can be improved. In [29], authors carried out a joint optimization study on task caching and offloading in the edge cloud environment, with the goal of reducing the energy consumption of task processing under computational and storage resource constraints. Following this, in [30], authors explored the challenges of dependency-aware task offloading and service caching within the context of vehicular networks. They devised a semi-distributed algorithm based on dynamic programming to enhance the efficiency of task offloading for vehicles.

Previous research has primarily focused on caching user tasks or related services on MEC servers. However, the issue of service placement has often been overlooked when deploying caching on servers. For example, In [31], the authors proposed a cooperative content placement problem aimed at minimizing the delivery delay of location-based content and the service cost of two types of content. In [32], authors focused on the joint optimization problem of MEC access network selection and service deployment. Their goal was to enhance the quality of service for user tasks in a cost-effective way by skillfully balancing access latency, communication latency, and service switching costs. In [33], to address the issue of service interruptions caused by mobile terminal devices roaming across different MEC server regions, an effective edge intelligent service placement algorithm was proposed. In [34], authors proposed a joint model for MEC server deployment and service placement, with the aim of maximizing the overall profit of all MEC servers while considering constraints on storage capacity and computing power.

However, these studies primarily concentrate on server deployment and service placement issues, rarely incorporating considerations for both service caching and task offloading.

2.3 Joint Service Caching and Computation Offloading in MEC

In [35], the authors developed an intelligent transportation system framework by integrating three computing layers: vehicles, network edge, and high-altitude platform stations. This framework considers the computation offloading strategy for vehicles, plans the caching strategy for network edge infrastructure data, and coordinates resource scheduling to enhance the delay performance of applications. In [36], a layered framework for edge intelligent IoT was constructed with computation offloading and content caching, aiming to minimize the total network latency of RSUs under long-term energy constraints. In [37], a study was conducted on an efficient caching scheme in Vehicle Edge Computing (VEC) based on edge-cloud collaboration. They proposed a VEC architecture utilizing Software-Defined Networking (SDN) with the goal of minimizing the migration latency of supporting data required for edge servers to execute tasks. In [38], the collaborative caching problem in edge environments was investigated with the aim of minimizing system costs, including data caching costs, data migration costs, and Quality of Service (QoS) losses. In [39], research was conducted on computational service caching in a multi-drone-assisted MEC system, addressing optimization problems related to joint service caching, task offloading, and drone placement. The study aims to minimize the overall delay of all devices while ensuring compliance with task delay requirements and energy budgets for both devices and drones. The aforementioned articles primarily address comprehensive considerations related to task offloading, service caching, and server placement. However, they seldom consider the integrated optimization of task offloading and service placement.

3 System Model and Problem Formulation

In this section, we will introduce the system model and give the problem formulation. The system model considered in this paper has a three-tier architecture, which includes a remote cloud server, multiple edge nodes (EN), and numerous users (as illustrated in Fig. 1). Assume that these users generate various types of task requests that need to be executed across the entire network. For each user's task request, the execution process may require different type of services, and these services are stored in ENs and the cloud server. Suppose the cloud server has stored all needed services, while in each EN, considering the limited storage resources, it only stores a small subset of these services. Suppose all ENs can communicate with each other and with the cloud server, while each user can only communicate with its local EN. This implies that for a user's request, the required services may not be stored on it's local EN. Therefore, the local EN needs to forward the request to other ENs for processing, or download the required services from the cloud server. In the following, we will proceed to introduce the network model, the service caching and placement model, the communication model, the computational model, and the system cost model respectively. Additionally, we will provide a detailed exposition of the multi-MEC server collaborative task processing problem based on service cache placement.

3.1 Network Model

We consider there are N ENs in the entire network model, and denote $\mathcal{N}$ as the set of ENs. For each $EN_j (\in \mathcal{N})$, it primarily consists with one wireless access point (AP) and one MEC server. The AP's job is for communicating, including communications with users, other APs, and the cloud. The MEC server's job is for computing, including computations for tasks and the storing for services. Specifically, let G_j denote the storage capacity of EN_j, which is used for storing the services required by tasks. When specific service caches are present in an EN, the EN can process tasks for users based on the corresponding service requests. Let B_j denote the available communication resources of EN_j, utilized for wireless data offloading of tasks. Let F_j denote the calculation capability of EN_j, which is used for processing tasks offloaded by users.

We consider there are M users in the entire network model, and denote $\mathcal{M}$ as the set of users. For each user $i (\in \mathcal{M})$, it may have a computationally intensive task needed to be processed during the scheduling time. Since each task has different resource requirements, four parameters are used to define the task model. Let U_i denote the task for user i, and we use the quadruple $\{D_i, C_i, K_i, T_i^{max}\}$ to denote resource requirements of U_i, where D_i is the workload data volume, C_i is the required computing resources, K_i is the set of required services, and T_i^{max} is the threshold of time latency.

3.2 Service Cache and Placement Model

We consider there are K services in the entire model, and denote $\mathcal{K}$ as the set of services. Let c_k denote the required storage space for service k. During the initialization phase, each EN downloads services from the cloud server. The storage capacity for

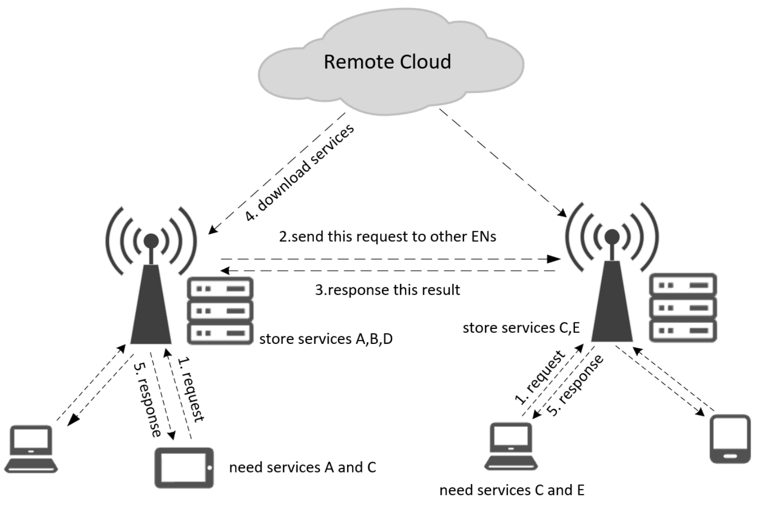

Fig. 1: Multi-ENs collaborative task processing model

services on each EN must not exceed the EN's capacity limit. Let $\mathcal{K}_j$ represent the service set of EN_j, we have

$$\sum_{k \in \mathcal{K}_j} c_k \leq G_j, \forall j \in \mathcal{N}. \tag{1}$$

Current research primarily employs two distinct approaches to solve the service placement problem. In one scenario, when an EN lacks the required services, it requests the corresponding service from other ENs and subsequently processes the task within its own environment. In another scenario, when an EN lacks the required services, it initiates a task request to other ENs and then processes the task within the environment of the other ENs. This paper comprehensively considers and improves upon two optimization approaches. Specifically, when a specific service is missing, in the first scenario, the service is downloaded by requesting the remote cloud server. In the second scenario, the local EN can initiate task requests to other ENs for processing. The optimal solution is designed for different scenarios to minimize system latency and energy consumption.

Let $x^t_{j,k} \in \{0,1\}$ denote a binary service placement decision of whether service k is deployed on EN_j in time slot t. $x^t_{j,k} = 1$ means service k is deployed on EN_j in time slot t, otherwise $x^t_{j,k} = 0$. For EN_j, the services it deploys can be represented by a collection. And we use $\{x^t_{j,1}, x^t_{j,2}, ..., x^t_{j,k}\}$ as the service resource deployed by EN_j.

3.3 Communication Model

In our scenario, we consider using Orthogonal Frequency Division Multiple Access (OFDMA) technology for wireless communication, which allows us to ignore interference between users. Let $R_{i,j}$ denote the uplink transmission rate between user i and

EN_j, we have

$$R_{i,j} = B_j log_2(1 + \frac{p_{i,j}h_{i,j}}{\sigma^2}), \forall i \in \mathcal{M}, j \in \mathcal{N}, \tag{2}$$

where B_j is the available spectrum bandwidth, $p_{i,j}$ is the uplink transmission power, σ^2 is the noise power and $h_{i,j}$ is the state of the uplink channel between user i and EN_j. Let $T_{i,j}^{up}$ denote the uplink transmission latency for task U_i to EN_j. It is primarily influenced by the allocated communication resources (i.e., wireless radio bandwidth resources) and the channel quality between user i and EN_j. The uplink transmission latency from task U_i to EN_j can be expressed as

$$T_{i,j}^{up} = \frac{D_i}{R_{i,j}}, \forall i \in \mathcal{M}, j \in \mathcal{N}. \tag{3}$$

Many studies indicate that the downlink transmission rate from the EN to the user is high, while the processed data volume is relatively small. Therefore, in our scheduling scenario, we only consider the transmission latency of the uplink link and ignore the latency of the return transmission.

As for the task request scheduling, we introduce a binary decision variable $y_{i,j}^t \in \{0, 1\}$ to denote whether task U_i is executed on local EN_j in time slot t. $y_{i,j}^t = 1$ means task U_i is executed on the local EN_j, otherwise $y_{i,j}^t = 0$. At a given time slot, since each user is handled by only one EN for the task, we have

$$\sum_{j \in \mathcal{N}} y_{i,j}^t = 1, \forall i \in \mathcal{M}. \tag{4}$$

Similarly, let the binary decision variable $z_{i,j,j'}^t \in \{0, 1\}$ to indicate whether task U_i is executed on other $EN_{j'}$ (i.e., $z_{i,j,j'}^t = 1$) or not (i.e., $z_{i,j,j'}^t = 0$) in time slot t. When the user i sends task request to EN_j, if EN_j doesn't have the required service k and EN_j needs to send a task request to $EN_{j'}$, we have $y_{i,j}^t = 0$ and $z_{i,j,j'}^t = 1$.

When EN_j executes task U_i, if the required service k is missing, the EN_j may initiate a task request to other ENs or download the service from the cloud server to EN_j. Let $R_{i,j,j'}^{round}$ denote the transmission rate of the task U_i between EN_j and $EN_{j'}$. Let $T_{i,j,j'}^{round}$ denote the round trip latency of task U_i between EN_j and $EN_{j'}$, we have

$$T_{i,j,j'}^{round} = \frac{D_i}{R_{i,j,j'}^{round}}, \forall i \in \mathcal{M}, j \in \mathcal{N}, j' \in \mathcal{N}. \tag{5}$$

Let $R_{c,j}^{down}$ denote the download rate between the cloud server and EN_j. Let $T_{c,j}^{down}$ denote the download latency for the required set of services from the cloud server to EN_j. We have

$$T_{c,j}^{down} = \sum_{k \in K_i} \frac{c_k}{R_{c,j}^{down}}, \forall j \in \mathcal{N}. \tag{6}$$

Thus the total transmission latency consists of three main components: uplink transmission latency $T_{i,j}^{up}$, round trip latency $T_{i,j,j'}^{round}$ and download latency $T_{c,j}^{down}$. Let

T_i^{trans} denote the total transmission latency, we have

$$\begin{aligned} T_i^{trans} =& T_{i,j}^{up} + \sum_{k \in K_i} y_{i,j}^t (1 - x_{j,k}^t) T_{c,j}^{down} \\ &+ \sum_{j' \in \mathcal{N}} z_{i,j,j'}^t (1 - y_{i,j}^t)(T_{i,j,j'}^{round} + \sum_{k \in K_i} (1 - x_{j',k}^t) T_{c,j'}^{down}). \end{aligned} \tag{7}$$

3.4 computational Model

Since the user's computational power is limited and the task is offloaded to the edge for execution, we can ignore the local computational latency. When user i sends task U_i to EN_j wirelessly, EN_j will allocate appropriate computational resources for user i to process the task. Eventually, after the task processing is completed at the edge end, EN returns the computational result to user i. Let $f_{i,j}^{mec}$ denote the computational resources allocated to user i by EN_j. Since the computational resources allocated by each EN to users must not exceed the total computational resources of that EN, each EN needs to satisfy

$$\sum_{i=1}^{M_j} f_{i,j}^{mec} \le F_j, \forall i \in \mathcal{M}, j \in \mathcal{N}, \tag{8}$$

where M_j is all users who execute tasks in EN_j. Let $T_{i,j}^{exe}$ denote the computational latency for EN_j to execute task U_i, we have

$$T_{i,j}^{exe} = \frac{C_i}{f_{i,j}^{mec}}, \forall i \in \mathcal{M}, j \in \mathcal{N}. \tag{9}$$

Let T_i^{exe} denote the computational latency for EN to execute task U_i, we have

$$T_i^{exe} = y_{i,j}^t T_{i,j}^{exe} + \sum_{j' \in \mathcal{N}} z_{i,j,j'}^t (1 - y_{i,j}^t) T_{i,j,j'}^{exe}. \tag{10}$$

Combining the transmission latency T_i^{trans} and the computational latency T_i^{exe}, the total latency of the EN execution task U_i is obtained as

$$T_i^t = T_i^{trans} + T_i^{exe}. \tag{11}$$

3.5 System Cost Model

There are additional costs involved in renting EN's resources to execute tasks, mainly including storage, communication and computation costs. Storage costs mainly include the overhead of using EN and remote cloud storage services, which mainly depends on the storage resources occupied by the services. Communication cost is the overhead incurred by data transfer between the user and the EN, which mainly depends on the amount of data transferred by the user. The computation cost is the overhead incurred by performing tasks in the EN, which mainly depends on the amount of data of the user's tasks. Note that we denote Q_i^{avg} as the long-term average system cost for user i. The system cost comprehensively considers expenses related to storage, communication, computation, and other factors. Let Q_i^t denote the system cost brought about by user i when renting EN resources during time slot t. We have

$$E_i^t = \sum_{k \in K_i} x_{j,k}^t \nu_j c_k + \sum_{k \in K_i} (1 - x_{j,k}^t) \mu_c c_k + \frac{\alpha_j p_{i,j} D_i}{B_j log_2(1 + \frac{p_{i,j} h_{i,j}}{\sigma^2})} + \beta_j C_i, \tag{12}$$

where ν_j and μ_c are the unit costs of storage services in EN_j and cloud servers, respectively. α_j and β_j are the unit costs of transmission energy consumption and task execution in EN_j, respectively.

3.6 Problem Formulation

In this section, we discuss the problem of collaborative task processing and resource allocation for multiple ENs based on service placement. Under the constraint of long-term average system cost, our goal is to minimize the total system latency for processing tasks. Therefore, we focus on the joint optimization of task offloading decisions, computational resources, and service placement decisions. Building upon the previous discussions, we can formulate the problem as follows:

$$\begin{aligned}
\mathbf{P1}: & \min_{x^t, y^t, z^t, f^t} \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} (\sum_{i=1}^{M} T_i^t) \\
\textbf{s.t. } & C1: \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} E_i^t \le E_i^{avg}, \quad \forall i \in \mathcal{M} \\
& C2: T_i^t \le T_i^{max}, \quad \forall i \in \mathcal{M} \\
& C3: \sum_{k \in \mathcal{K}_j} c_k \le G_j, \quad \forall j \in \mathcal{N} \\
& C4: f_{i,j}^{mec} > 0, \quad \forall i \in \mathcal{M}, j \in \mathcal{N} \\
& C5: \sum_{i=1}^{M_j} f_{i,j}^{mec} \le F_j, \quad \forall i \in \mathcal{M}, j \in \mathcal{N} \\
& C6: x_{j,k}^t \in \{0, 1\}, \quad \forall j \in \mathcal{N}, k \in \mathcal{K} \\
& C7: y_{i,j}^t \in \{0, 1\}, \quad \forall i \in \mathcal{M}, j \in \mathcal{N} \\
& C8: \sum_{j \in \mathcal{N}} y_{i,j}^t = 1, \quad \forall i \in \mathcal{M} \\
& C9: z_{i,j,j'}^t \in \{0, 1\}, \quad \forall i \in \mathcal{M}, j \in \mathcal{N}, j' \in \mathcal{N}
\end{aligned} \tag{13}$$

C1 represents the long-term average system cost threshold for each user. C2 is the maximum limit on the total latency of executing the task. C3 indicates that EN's storage space is limited. C4 is the computational resource constraint assigned by EN to the task. C5 means that the computational resources allocated by EN to the task do not exceed EN's computational capabilities. C6 is a constraint on the binary decision variable for the existence of service k in EN_j. C7 and C9 are constraints on the binary offloading decision variables, which determine whether the task is executed on the local EN or $EN_{j'}$ respectively. C8 ensures that tasks can only be executed on one EN.

Solving the above-mentioned problem primarily involves addressing two challenges. Firstly, resolving P1 necessitates a significant amount of future information, which is often difficult to predict, and in some cases, even impossible. Secondly, we have identified that P1 is a mixed-integer nonlinear programming problem. It encompasses constraints related to three discrete binary variables and continuous variables C4-C5, rendering it an NP-hard problem. This indicates a high degree of computational complexity, making a direct solution to P1 evidently impractical. To overcome these challenges, we need to devise an online algorithm capable of effectively addressing optimization issues related to offloading decisions, computational resource allocation, and service placement without relying on future information.

4 Our Proposed Online Algorithm

To effectively solve problem P1, we propose an online algorithm named OPDA that does not require future information. Our algorithm is primarily based on the Lyapunov optimization framework, which transforms the original long-term optimization problem P1 into multiple one-slot optimization problems. For the one-slot optimization problem, we adopt an algorithm based on the Lagrange multiplier method and matching theory to determine the optimal offloading decision, computing resource allocation, and service placement problem.

4.1 Problem Transformation

In this section, we will restate and transform problem P1. The core idea of Lyapunov optimization is to make decisions that approach optimality while ensuring the stability of the system, allowing us to solve deterministic problems in each time slot with low complexity[40]. Based on this concept, we can design a set of virtual cost queues $Q(t) = \{Q_1(t), Q_2(t), ..., Q_i(t)\}$ to address constraints on the long-term average cost budget. The virtual cost queue for user i can be expressed as

$$Q_i(t+1) = max\{Q_i(t) + E_i^t - E_i^{avg}, 0\}, \tag{14}$$

where $Q_i(t)$ denotes the queue backlog of time slot t, representing the deviation between the system cost and the threshold under the current time slot t. Additionally, the initial queue backlog is given by $Q(0) = 0$. According to queueing theory, C1 can only be satisfied when the virtual cost queue remains stable, meaning that the average arrival rate does not exceed the average departure rate. Therefore, ensuring the stability of $Q(t)$ becomes a crucial factor.

To maintain the stability of $Q(t)$, we denote $L(Q(t)) = \frac{1}{2}\sum_{i=1}^{M} Q_i^2(t)$ as a quadratic Lyapunov function, representing the "congestion level" of the virtual cost queue length. A smaller value of $L(Q(t))$ indicates a smaller backlog for all virtual cost queues, thereby achieving stability for all queues. Next, we further denote $\Delta(Q(t))$ as a single time slot Lyapunov drift to ensure that the Lyapunov function remains at a low value, thereby preserving the stability of the queues. we have

$$\Delta(Q(t)) = \mathbb{E}[L(Q(t+1)) - L(Q(t))|Q(t)], \tag{15}$$

where $\Delta(Q(t))$ denotes the variation of the virtual cost queue in the Lyapunov function over a time slot. Based on the Lyapunov optimization framework, we further define a drift-plus-penalty function to integrate the stability of the system latency and virtual cost queue by (15).

$$\Delta(Q(t)) + V\mathbb{E}[\sum_{i=1}^{M} T_i^t|Q(t)], \tag{16}$$

where V is a non-negative parameter that controls the trade-off between system latency and cost. The following **Lemma 1** provides an upper bound on the drift-plus-penalty function.

Lemma 1: For all possible optimization variables satisfying C2-C9, there is an upper bound for the drift-plus-penalty function [41], as shown in (17). Where B is a constant, $B = \frac{1}{2}\sum_{i=1}^{M}(E_i^{max} - E_i^{avg})^2$. Where E_i^{max} is the upper bound of the one time slot virtual cost for user i.

$$\Delta(Q(t)) + V\mathbb{E}[\sum_{i=1}^{M} T_i^t|Q(t)] \leq B + V\mathbb{E}[\sum_{i=1}^{M} T_i^t|Q(t)] + \mathbb{E}[\sum_{i=1}^{M} Q_i(t)(E_i^t - E_i^{avg})|Q(t)] \tag{17}$$

Proof: By transforming formula (14), we can know,

$$Q_i^2(t+1) \leq (Q_i(t) + (E_i^t - E_i^{avg}))^2 \tag{18}$$

Combined with our analysis, all queues are accumulated on the basis of formula (18), and it can be concluded that

$$\frac{1}{2}\sum_{i=1}^{M} Q_i^2(t+1) \leq \frac{1}{2}\sum_{i=1}^{M} Q_i^2(t) + \frac{1}{2}\sum_{i=1}^{M}(E_i^t - E_i^{avg})^2 + \sum_{i=1}^{M} Q_i(t)(E_i^t - E_i^{avg}). \tag{19}$$

Therefore it can be concluded that

$$\begin{aligned} \Delta(Q(t)) &= \frac{1}{2}\mathbb{E}[\sum_{i=1}^{M} Q_i^2(t+1) - \frac{1}{2}\sum_{i=1}^{M} Q_i^2(t)|Q(t)] \\ &\leq \frac{1}{2}\sum_{i=1}^{M}(E_i^t - E_i^{avg})^2 + \mathbb{E}[\sum_{i=1}^{M} Q_i(t)(E_i^t - E_i^{avg})|Q(t)] \\ &\leq B + \mathbb{E}[\sum_{i=1}^{M} Q_i(t)(E_i^t - E_i^{avg})|Q(t)]. \end{aligned} \tag{20}$$

Proof completed.

As demonstrated in Lemma 1, we observe that by minimizing the expression on the right-hand side of formula (17), it is possible to determine the upper bound of the drift-plus-penalty function. This will help us transform the long-term optimization problem P1 into a real-time optimization problem P2. By solving P2, we can ascertain the offloading decisions, resource allocation, and the placement strategy for service in the current time slot.

$$\mathbf{P2}: \min_{x^t,y^t,z^t,f^t} \sum_{i=1}^{M} VT_i^t + Q_i(t)E_i^t \tag{21}$$

In Algorithm 1, we provide a detailed description of the implementation process of OPDA. The algorithm requires current available network state information and the backlog status of the current queue as input, making OPDA operate online. In each time slot t, by solving P2, we can obtain the values of variables x^t, y^t, z^t and f^t, and subsequently update each virtual cost queue for the calculation of the next time slot. We can obtain the optimal solution to problem P1 by solving the approximate optimal solution to problem P2.

Algorithm 1 OPDA Algorithm

```
Require: All user set information $\mathcal{M}$, all ENs set information $\mathcal{N}$, user task $U_i$, all
    services set information $\mathcal{K}$ and non-negative parameter V
Ensure: The offloading decision $y^t, z^t$, the computational resource $f^t$, and the
    placement decision $x^t$ of the caching service within each time slot
1: while t = 0 to T-1 do
2:     According to Algorithm 2 and Algorithm 3, we can obtain the values of $x^t, y^t, z^t$
       and $f^t$ for each time slot.
3:     Update each virtual cost queue Q(t+1) according to (14).
4:     t = t + 1.
5: end while
```

Although transforming the long-term optimization problem into a real-time optimization problem, it can be seen that solving P2 using mathematical methods is quite challenging, as shown in the formula (21) above. Firstly, solving problem P2 has exponential complexity. Secondly, the variables x^t, y^t and z^t in P2 are binary variables, while f^t is a continuous variable. Therefore, P2 is categorized as a mixed-integer nonlinear programming problem, typically falling into the category of NP-hard problems[42]. Subsequent steps to solve the real-time optimization problem P2 will be discussed in the next section.

4.2 One time slot optimization algorithm

To address problem P2, we can decompose it into two subproblems for resolution: given the conditions of x^t, y^t and z^t, solving the allocation problem of computing resources f^t using the Karush-Kuhn-Tucker (KKT) optimality conditions. Based on the obtained value of computational resources f^t, subsequently, through a bilateral matching of user task set and EN set, as well as the matching between the services and the set of ENs, we derive the values of x^t, y^t and z^t. By iteratively solving these two subproblems, we can progressively optimize problem P2 and ultimately find the optimal solution.

4.2.1 The Optimal Computing Resource Allocation.

In this section, to maximize system utility for users offloading tasks to the edge, we strive to allocate optimal computing resources for them. Assuming x^t, y^t and z^t are given, we will ignore variables unrelated to the computing resource f^t. Consequently, the optimization problem can be formulated as follows:

$$\begin{aligned} &\textbf{P3}: \min_{f^t} \sum_{i=1}^{M} \frac{C_i}{f_{i,j}^{mec}} \\ &\textbf{s.t. } C4: f_{i,j}^{mec} > 0, \quad \forall i \in \mathcal{M}, j \in \mathcal{N} \\ &\qquad C5: \sum_{i=1}^{M_j} f_{i,j}^{mec} \leq F_j, \quad \forall i \in \mathcal{M}, j \in \mathcal{N} \end{aligned} \tag{22}$$

Let $G_1(f_{i,j}^{mec})$ be designated as the objective function for problem P2. In order to prove that P3 is a convex problem, we take the second derivative of $G_1(f_{i,j}^{mec})$ with respect to $f_{i,j}^{mec}$. we have

$$\frac{\partial^2 G_1(f_{i,j}^{mec})}{\partial (f_{i,j}^{mec})^2} = \frac{2C_i}{(f_{i,j}^{mec})^3}. \tag{23}$$

Due to constraint C4, we have $\frac{\partial^2 G_1(f_{i,j}^{mec})}{\partial (f_{i,j}^{mec})^2} > 0$. Furthermore, considering that constraint C4 is linear, and f^t is a continuous variable, the problem presented in Equation (22) is convex. Given that the problem is convex and satisfies the Slater condition, to obtain the optimal solution for (22), we can use the Lagrangian function to solve P3. We have

$$L(f, \lambda) = \sum_{i=1}^{M} \frac{C_i}{f_{i,j}^{mec}} + \lambda\left(\sum_{i=1}^{M_j} f_{i,j}^{mec} - F_j\right), \tag{24}$$

where λ is a non-negative Lagrangian multiplier associated with the computing resource constraints of EN. Then, we use the KKT conditions to obtain the optimal allocation f^t for computational resources[43]. By taking the partial derivative of $L(f, \lambda)$ with respect to $f_{i,j}^{mec}$ and setting it equal to 0, we can obtain

$$\frac{\partial L(f, \lambda)}{\partial f_{i,j}^{mec}} = -\frac{C_i}{(f_{i,j}^{mec})^2} + \lambda. \tag{25}$$

Subsequently, utilizing the binary search method, we can obtain an approximate optimal solution for equation (25). We have

$$f^* = \sqrt{\frac{C_i}{\lambda}}. \tag{26}$$

Then we can obtain the optimal computing resource allocation scheme f^* based on Algorithm 2. Algorithm 2 provides a detailed description of the specific process for obtaining the optimal computational resource allocation.

Algorithm 2 Optimal Computing Resource Allocation Algorithm

```
Require: All user set information $\mathcal{M}$, all ENs set information $\mathcal{N}$, user task $U_i$,
    Maximum tolerance $\zeta = 1 \times 10^{-8}$, $\lambda_{min} = 0$ and $\lambda_{max} = 1$
Ensure: The optimal computing resource $f^*$
1: for all users $i \in \mathcal{M}$ do
2:     while $\lambda_{max} - \lambda_{min} > \zeta$ do
3:         Get the value of $\lambda$ according to the formula $\lambda = (\lambda_{max} + \lambda_{min})/2$.
4:         Calculate $f^*$ by substituting the obtained value of $\lambda$ into equation (26).
5:         If constraint C5 is satisfied, we will update $\lambda_{max} = \lambda$.
6:         Otherwise, we will update $\lambda_{min} = \lambda$.
7:     end whileThrough formula (26), we can get the optimal computing resources
    $f^*$ for each user.
8: end for
```

4.2.2 The Optimal Task Offloading and Service Placement Decision.

From the preceding discussion,it can be inferred that the primary challenge in addressing this issue lies in the high coupling between task offloading decisions and service placement decisions. Moving forward, we should reframe problem P2 into one that focuses on optimizing task offloading decisions and service placement decisions to reduce system latency and costs when the computing resource f^* is determined. Therefore, we can formulate the problem as follows:

$$\begin{aligned} &\mathbf{P4}: \min_{x^t, y^t, z^t} G_2(x^t_{j,k}, y^t_{i,j}, z^t_{i,j,j'}) \\ &\textbf{s.t. } C2 - C9 \end{aligned} \tag{27}$$

$$\begin{aligned} G_2(x^t_{j,k}, y^t_{i,j}, z^t_{i,j,j'}) = &\sum_{i=1}^{M}\sum_{j=1}^{N} V[T^{up}_{i,j} + y^t_{i,j}(\frac{C_i}{f^*_{i,j}} + \sum_{k\in K_i}(1 - x^t_{j,k})T^{down}_{c,j}) \\ &+ \sum_{j'\in\mathcal{N}} z^t_{i,j,j'}(1 - y^t_{i,j})(\frac{C_i}{f^*_{i,j'}} + T^{round}_{i,j,j'} + \sum_{k\in K_i}(1 - x^t_{j',k})T^{down}_{c,j'})] + Q_i(t)E^t_i \end{aligned} \tag{28}$$

From a macro perspective, the offloading of user tasks and the placement of services can be seen as a variant of the stable matching problem, akin to the college admissions problem (CAP). Unlike traditional one-to-one matching, CAP aims for stable many-to-one bilateral matching, where a specific college can be open to and admit multiple students. Similarly, a student can choose multiple colleges but can ultimately select only one. In our context, each EN corresponds to a college, and users and cache services play the roles of students in the CAP framework. According to the concept of stable matching theory, it is essential to precisely define the interests and preferences of two participants. For example, we can use "$a_c > b_c$" to indicate that c prefers a over b. Therefore, it is necessary to identify the specific interests of each participant in the problem and create preference lists for them based on these interests. To address P4, we propose a task offloading and service placement algorithm based on matching theory, striving to find a stable optimal match between the sets of tasks and ENs, as well as between the sets of services and ENs[44]. This ensures that tasks achieve maximum system utility in the current matching.

In this paper, users tend to offload tasks to EN that have the required services and sufficient computing resources, while ENs are more inclined to handle tasks with low computational resource requirements. Similarly, services tend to store data on ENs that handle relevant tasks, and ENs tend to store fewer services to reduce the overall system costs. For simplification, we denote $\mathcal{U}$ as a set of tasks and $\mathcal{N}$ as a set of ENs, which are two disjoint participants. We then perform the matching between $\mathcal{U}$ and $\mathcal{N}$. Each task can only be assigned to one EN for processing, but each EN can handle multiple tasks. W denote $H_1(x)$ as the matching function between $U_i \in \mathcal{U}$ and $EN_j \in \mathcal{N}$, which is a mapping from the set $\mathcal{U} \cup \mathcal{N}$ to the subset of $\mathcal{U} \cup \mathcal{N}$. It needs to satisfy the following three conditions: (1). $H_1(U_i) \in \mathcal{N}, \forall U_i \in \mathcal{U}$ means that any task can be offloaded and processed on any EN in the set $\mathcal{N}$. (2). $H_1(EN_j) \in \mathcal{U}, \forall j \in \mathcal{N}$ indicates that any task processed by EN in set $\mathcal{N}$ is included in set $\mathcal{U}$. (3). $H_1(EN_j) = U_i \Leftrightarrow H_1(U_i) \in EN_j$ implies that the task U_i to EN_j processing is equivalent to EN_j processing of tasks including U_i, but not only processing a task U_i. If there is a current matching $H_1(x)$, the offloading decision of the task can be determined as

$$y_{i,j}^t = \begin{cases} 1\ , & \text{if } H_1(EN_j) = U_i; \\ 0\ , & \text{otherwise.} \end{cases} \tag{29}$$

$$z_{i,j,j'}^t = \begin{cases} 1\ , & \text{if } H_1(EN_{j'}) = U_i \textit{ and } y_{i,j}^t = 0,\ j \neq j'; \\ 0\ , & \text{otherwise.} \end{cases} \tag{30}$$

Each participant can establish a preference list with other participants by formulating a utility function[45], which is defined as inversely proportional to the sum of the total system latency and cost. Therefore, the utility function for task U_i can be expressed as

$$SU_{U_i}(EN_j) = \frac{1}{G_2(y_{i,j}^t, z_{i,j,j'}^t)}. \tag{31}$$

If there is the following definition $SU_{U_i}(EN_{j_1}) > SU_{U_i}(EN_{j_2})$, it means that the task U_i prefers to be processed in EN_{j_1} rather than in EN_{j_2}.

Similarly, we denote $\mathcal{K}$ as a set of services and denote $\mathcal{N}$ as a set of ENs, which are two disjoint participants. Services can only be selected for storage in one EN, and multiple services can be stored in each EN. Denote $H_2(x)$ as the matching function between $k \in \mathcal{K}$ and $EN_j \in \mathcal{N}$. If a match currently exists for $H_2(x)$, then the service placement decision can be determined as

$$x_{j,k}^t = \begin{cases} 1\ , & \text{if } H_2(EN_j) = k; \\ 0\ , & \text{otherwise.} \end{cases} \tag{32}$$

Then the utility function of EN for task U_i can be expressed as

$$QU_k(EN_j) = \frac{1}{G_2(x_{j,k}^t)}. \tag{33}$$

If the definition states that $QU_k(EN_{j_1}) > QU_k(EN_{j_2})$, this implies that the service k is more inclined to be stored in EN_{j_1} rather than in EN_{j_2}.

Algorithm 3 addresses the specific process of solving problem P4 in two main phases: an initialization phase and a matching phase. In the initialization phase, all users offload tasks to ENs while storing services in random ENs and creating corresponding matches relationships. In the matching phase, users seek the corresponding ENs for task processing, and cache services also search for the corresponding ENs for storage, establishing stable matching relationships. If the system utility of task U_i with the currently matched EN is low, user i will send a matching request to other ENs. Similarly, if the current matching with the EN cache associated with this service results in additional high costs and prolonged task processing latency, the service may choose to send matching requests to other ENs.

When other ENs receive matching requests for tasks, they calculate the system utility and make matching decisions: (1) If accepting task U_i can improve the system utility, they accept the matching request from user i. (2) Otherwise, they reject the matching request from user i. If the matching request from user i is rejected, user i will send a matching request to the next available EN. Likewise, when other ENs receive matching requests for services, they also calculate additional costs and task processing latency and make matching decisions: (1) If the service cache in that EN can reduce additional costs and decrease task processing latency, they accept the matching request from service k. (2) Otherwise, they reject the matching request from service k. If the matching request from service k is rejected, service k will send a matching request to the next available EN.

When there are no matching requests from users and services during the matching phase, the matching phase ends. In each round of the matching process between tasks and cache services with ENs, stable matching is achieved on both sides, maximizing system utility. Ultimately, based on the matching results that satisfy all constraints, optimal offloading decisions for all tasks and optimal placement decisions for cache services can be determined.

Algorithm 3 Task Offloading and Service Placement Algorithms Based on Matching Theory

```
Require: All user set information $\mathcal{M}$, all ENs set information $\mathcal{N}$, user task $U_i$ and all services set information $\mathcal{K}$
Ensure: The placement decision $x^t$ for the service, and the optimal offloading decisions $y^t$ and $z^t$ for all tasks
 1: Initialize Phase:
 2: Match all user tasks with local EN while satisfying constraints C2-C9.
 3: Randomly deploying all services on EN.
 4: Calculate the utility under the current matching according to (31) and (33).
 5: Matching Phase:
 6: while there exists unstable matching do
 7:     if current EN does not have the services required for the user task then
 8:         Consider the following two scenarios.
 9:         1. Query $EN_{j'}$ with the services required by the user.
10:         Retrieve all parameter information for $EN_{j'}$.
11:         Calculate the system utility of user $i$ to $EN_{j'}$.
12:         for all ENs $j' \in \mathcal{N}$ and $j' \neq j$ do
13:             If $SU_{U_i}(EN_j) < SU_{U_i}(EN_{j'})$
14:             $EN_{j'}$ will accept the match request and offloads the task to $EN_{j'}$.
15:             Otherwise, tasks are offloaded to the current $EN_j$ for processing.
16:         end for
17:         2. EN stores only the services required by the current user.
18:         Remove services that are not currently needed.
19:         Calculate system utility of service $k$ to $EN_{j'}$ while satisfying constraints.
20:         for all ENs $j' \in \mathcal{N}$ and $j' \neq j$ do
21:             If $QU_k(EN_j) < QU_k(EN_{j'})$, service k will be stored in $EN_{j'}$.
22:             Otherwise, service $k$ is still stored in this current EN.
23:         end for
24:     end if
25: end while
26: The system utility no longer changes, i.e., a stable match $H$ is reached between tasks and ENs, and a stable match $G$ is formed between services and ENs.
27: Base on (29), (30), (32) and the matching results, we can get the placement decision of the service $x_t$, and the optimal offloading decision of all tasks $y^t, z^t$.
```

5 Simulation Results

In this section, we will evaluate our proposed OPDA algorithm through experimental simulation results. We will compare the OPDA algorithm with other algorithms and conduct an in-depth analysis of the comparison results.

5.1 Simulation Setting

In the scenario of multi-ENs cooperative task processing and resource allocation based on service placement, we can set a series of parameters for simulation experiments. Firstly, we can set a coverage area of 500m $\times$ 500m for the scenario, which includes 10 users and 4 ENs, all randomly distributed within the coverage area. Secondly, we can set the transmission power $p_{i,j}$ of user i to 30dBm[46], and the noise power σ^2 is set to -100dBm. At the same time, we have set the wireless channel gain $h_{i,j}$ from user i to EN_j as $h_{i,j} = d_{i,j}^{-\vartheta}$, where $d_{i,j}$ represents the distance between user i and the relevant EN_j, and the path loss factor $\vartheta = 4$. Considering the heterogeneity of ENs and user tasks, we set the input data size D_i of each user task to be within the range of [30, 50]MB[47], and the CPU cycles required to execute the task is 1200. Next, we set the channel bandwidth B_j, storage capacity G_j, and computing resources F_j of EN_j to be [8, 12]MHz, [5, 10]GB, and [30, 60]GHz, respectively. In order to more effectively handle tasks offloaded by users to the edge end, we must occupy various resources of the EN, which will also bring additional costs. Therefore, we set the transmission energy cost α_j of the task and the unit cost β_j of executing the task to be $[2 \times 10^{-4}, 3 \times 10^{-4}]$ unit/J and $[0.4 \times 10^{-7}, 0.8 \times 10^{-7}]$ units/bit, respectively. Finally, we set the unit costs of the remote cloud server and the storage service of the EN to be [1.0, 1.2] units and [0.54, 0.6] units[48], respectively. These settings will help us to simulate and evaluate the performance of the algorithm more accurately.

Next, we conduct a performance study of the proposed OPDA algorithm and compared it with other algorithms. We evaluate the OPDA algorithm using 500 time slots, with a time interval of 10 ms for each slot. Additionally, to assess the performance of the OPDA algorithm, we introduce four baseline algorithms: (1). Independent EN Processing Algorithm(IEPA): In this algorithm, ENs within the coverage area independently handle tasks without any collaborative relationship. Tasks can only be processed locally in the respective ENs, with each EN allocating appropriate computing resources. (2). Optimal System Latency Processing Algorithm(OSLPA): When processing tasks, this algorithm maximizes efforts to reduce system latency without considering system cost constraints. (3). Random Processing Algorithm: Due to mutual communication among ENs, tasks can be randomly offloaded to any EN for processing. In these three algorithms, there is no consideration for the placement of service caches; services are randomly deployed on ENs. (4). Service Cache Random Placement Algorithm(SCRPA): This algorithm disregards service cache placement issues but maintains consistency with this paper regarding task offloading and resource allocation problems.

5.2 The Effect of Parameter V on System Performance

Fig. 2 and 3 illustrate the average system latency and cost when employing OPDA and OSLPA algorithms to handle 10 tasks under different parameters V. From Fig. 2 it can be observed that as the parameter V increases, the average system latency of the OPDA algorithm significantly decreases, gradually approaching the optimal processing latency of the system. Similarly, Fig. 3 reveals that with the increase in parameter

V, the average system cost of the OPDA algorithm significantly rises, ultimately maintaining a level below the system cost of the OSLPA algorithm. This indicates that the OSLPA algorithm achieves the best system processing latency at the expense of higher system costs.

These two figures validate that the parameter V can achieve a controllable balance between system latency and cost, exhibiting a relationship of [O(1/V), O(V)], consistent with Lyapunov's characteristics. As the non-negative parameter V increases, the system's latency performance decreases, but the system cost simultaneously increases. Therefore, we can adjust the parameter V based on specific circumstances to meet the diverse requirements of the system. Even in the absence of network status information, optimizing the parameter V allows us to attain optimal system latency performance and cost. Considering the constraints of long-term costs, our algorithm demonstrates a clear advantage over the OSDPA algorithm. This advantage is not only reflected in the improvement of system latency performance but also in the more effective control of system costs.

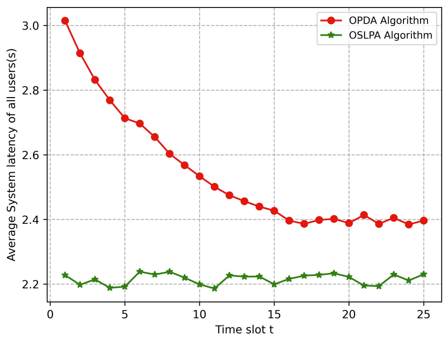

Fig. 2: The impact of different V

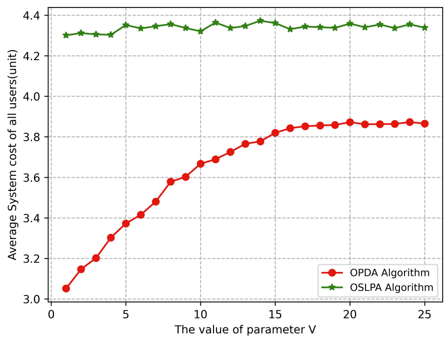

Fig. 3: The impact of different V

5.3 Performance Comparison

Based on the trade-off analysis of system latency and cost, a larger V value will make the system pay more attention to latency performance, but it may violate the long-term average system cost within a limited time slot. Therefore, we show the average latency and cost performance of 4 ENs processing 10 tasks under the condition of V = 4 in Fig. 4 and 5. These two figures clearly show that the OPDA algorithm achieves nearly optimal latency performance while satisfying the long-term cost constraint. Although the OSLPA algorithm performs best in terms of latency performance, Fig. 5 clearly shows that using the OSLPA algorithm requires a large amount of system cost. In contrast, the OPDA algorithm slightly sacrifices some latency performance to meet the long-term average cost constraint. Since the OPDA algorithm adds a constraint of long-term average system cost in each time slot, our algorithm also meets the constraint of long-term average system cost.

On the other hand, because ENs can cooperate with each other in communication, the random algorithm can randomly assign tasks to any EN without considering specific resource requirements, resulting in poor latency and cost-effectiveness. In addition, in the IEPA algorithm, each EN processes the task requests sent by users independently, without considering the cooperative relationship between ENs, resulting in poor overall system latency performance. In general, compared with the other three algorithms, the system latency of our proposed OPDA algorithm is lower, and the long-term average system cost also maintains a lower level.

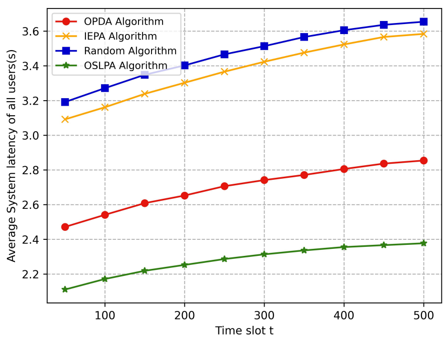

Fig. 4: Average system latency

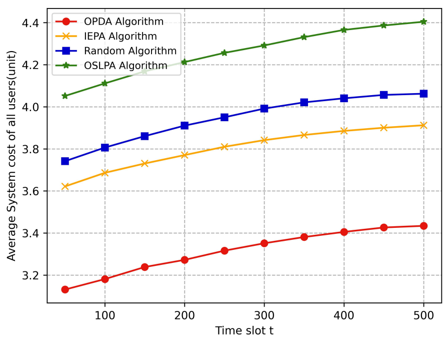

Fig. 5: Average system cost

5.4 The Effect of The Number of Users on System Performance

Fig. 6 and 7 respectively reveal the total system latency and cost when we handle different numbers of tasks using different schemes. As the number of tasks increases, the system latency and cost in all scenarios rise. Among them, the OSLPA algorithm achieves the lowest system latency, followed closely by the OPDA algorithm, while the IEPA and random algorithms have the highest system latency.

Combining Fig. 6 and 7, although the OSLPA algorithm maintains a relatively low system processing latency, its system cost for handling tasks is the highest. This indicates that the algorithm consumes a large amount of system cost to achieve the best system latency, which obviously does not meet our expectations. In addition, we can also observe that compared with other algorithms, the OPDA algorithm maintains the lowest system cost when handling tasks, and the system latency also remains at a relatively low level. In summary, the algorithm we propose can effectively handle user tasks, maintaining a relatively low long-term average system cost while its system processing latency is closer to the optimal state.

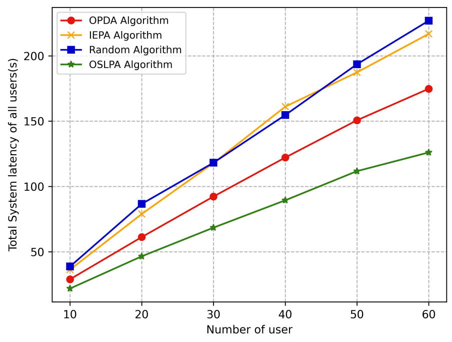

Fig. 6: System latency with different M

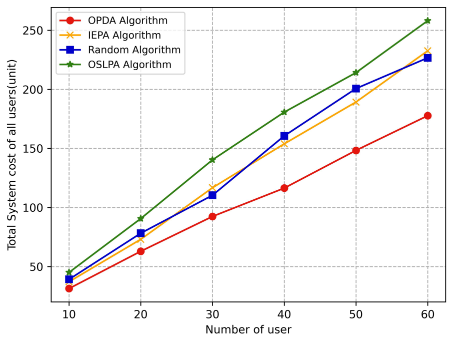

Fig. 7: System cost with different M

5.5 The Effect of Service Placement on System Performance

Fig. 8 and 9 illustrate the impact of employing OPDA and SCRPA algorithms on system latency performance and costs. As the number of tasks increases, both algorithms exhibit a rising trend in system latency and costs. However, in comparison to the SCRPA algorithm, the use of OPDA consistently maintains lower levels of system latency and costs. In the absence of considering the optimization of service placement, services can be arbitrarily placed on any given EN. As seen in Fig. 8, this may result in the current EN not storing the necessary services for tasks, requiring the download of these services from cloud servers, thereby increasing the processing latency of EN for tasks. On the other hand, as observed in Fig. 9, this could also lead to the current EN storing many services that tasks do not require, wasting a significant amount of storage resources and consequently increasing the storage costs for EN in handling tasks. In summary, this indicates that our OPDA algorithm is highly effective. By optimizing the service placement issue, it significantly reduces latency and additional costs incurred during task processing at the edge.

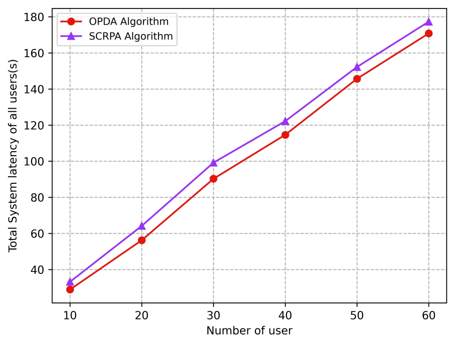

Fig. 8: System latency with different M

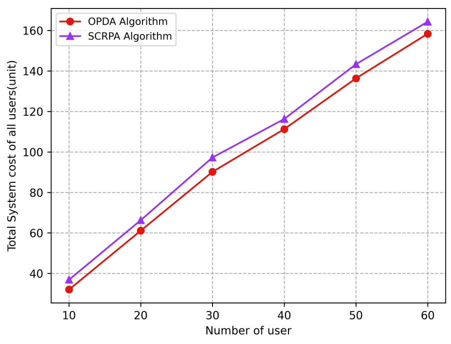

Fig. 9: System cost with different M

6 Conclusion

In this paper, we propose a multi-ENs collaborative task processing model based on service cache placement and formulate a MINLP optimization problem aimed at minimizing system latency and cost. The main research content is to store the services required by users in the EN and to collaboratively process tasks offloaded to the edge end by users among multiple ENs. In this process, we make optimal offloading decisions and compute resource allocations for tasks, while optimizing the placement decisions of services. To solve this problem, we propose an online optimization algorithm (OPDA), which operates online and does not predict future information. We then decompose the long-term optimization problem into a series of one-time slot problems and design a two-stage one-time slot optimization algorithm to obtain an approximate optimal solution. Finally, simulation results show that this algorithm can achieve near-optimal latency performance under the premise of satisfying long-term cost constraints.

References

[1] I. A. Elgendy, W. -Z. Zhang, Y. Zeng, H. He, Y. -C. Tian and Y. Yang, "Efficient and Secure Multi-User Multi-Task Computation Offloading for Mobile-Edge Computing in Mobile IoT Networks," in IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 2410-2422, Dec. 2020.

[2] E. El Haber, T. M. Nguyen and C. Assi, "Joint Optimization of Computational Cost and Devices Energy for Task Offloading in Multi-Tier Edge-Clouds," in IEEE Transactions on Communications, vol. 67, no. 5, pp. 3407-3421, May 2019.

[3] M. Zhao et al., "Energy-Aware Task Offloading and Resource Allocation for Time-Sensitive Services in Mobile Edge Computing Systems," in IEEE Transactions on Vehicular Technology, vol. 70, no. 10, pp. 10925-10940, Oct. 2021.

[4] Q. Li, S. Wang, A. Zhou, X. Ma, F. Yang and A. X. Liu, "QoS Driven Task Offloading With Statistical Guarantee in Mobile Edge Computing," in IEEE Transactions on Mobile Computing, vol. 21, no. 1, pp. 278-290, 1 Jan. 2022.

[5] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu and X. S. Shen, "TOFFEE: Task Offloading and Frequency Scaling for Energy Efficiency of Mobile Devices in Mobile Edge Computing," in IEEE Transactions on Cloud Computing, vol. 9, no. 4, pp. 1634-1644, 1 Oct.-Dec. 2021.

[6] T. Zhou, Y. Yue, D. Qin, X. Nie, X. Li and C. Li, "Mobile Device Association and Resource Allocation in HCNs With Mobile Edge Computing and Caching," in IEEE Systems Journal, vol. 17, no. 1, pp. 976-987, March 2023.

[7] S. Raza, S. Wang, M. Ahmed, M. R. Anwar, M. A. Mirza and W. U. Khan, "Task Offloading and Resource Allocation for IoV Using 5G NR-V2X Communication," in IEEE Internet of Things Journal, vol. 9, no. 13, pp. 10397-10410, 1 July1, 2022.

[8] J. Zhou and X. Zhang, "Fairness-Aware Task Offloading and Resource Allocation in Cooperative Mobile-Edge Computing," in IEEE Internet of Things Journal, vol. 9, no. 5, pp. 3812-3824, 1 March1, 2022.

[9] X. Xia et al., "OL-MEDC: An Online Approach for Cost-Effective Data Caching in Mobile Edge Computing Systems," in IEEE Transactions on Mobile Computing, vol. 22, no. 3, pp. 1646-1658, 1 March 2023.

[10] F. Zhang, G. Han, L. Liu, M. Mart??nez-Garc??a and Y. Peng, "Joint Optimization of Cooperative Edge Caching and Radio Resource Allocation in 5G-Enabled Massive IoT Networks," in IEEE Internet of Things Journal, vol. 8, no. 18, pp. 14156-14170, 15 Sept.15, 2021.

[11] C. Song, W. Xu, T. Wu, S. Yu, P. Zeng and N. Zhang, "QoE-Driven Edge Caching in Vehicle Networks Based on Deep Reinforcement Learning," in IEEE Transactions on Vehicular Technology, vol. 70, no. 6, pp. 5286-5295, June 2021.

[12] Z. Li, X. Gao, Q. Li, J. Guo and B. Yang, "Edge Caching Enhancement for Industrial Internet: A Recommendation-Aided Approach," in IEEE Internet of Things Journal, vol. 9, no. 18, pp. 16941-16952, 15 Sept.15, 2022.

[13] J. Zhang, H. Guo, J. Liu and Y. Zhang, "Task Offloading in Vehicular Edge Computing Networks: A Load-Balancing Solution," in IEEE Transactions on Vehicular Technology, vol. 69, no. 2, pp. 2092-2104, Feb. 2020.

[14] J. Chen, H. Wu, P. Yang, F. Lyu and X. Shen, "Cooperative Edge Caching With Location-Based and Popular Contents for Vehicular Networks," in IEEE Transactions on Vehicular Technology, vol. 69, no. 9, pp. 10291-10305, Sept. 2020.

[15] Y. Gao and G. Casale, "JCSP: Joint Caching and Service Placement for Edge Computing Systems," 2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS), Oslo, Norway, 2022, pp. 1-10.

[16] C. Kai, H. Zhou, Y. Yi and W. Huang, "Collaborative Cloud-Edge-End Task Offloading in Mobile-Edge Computing Networks With Limited Communication Capability," in IEEE Transactions on Cognitive Communications and Networking, vol. 7, no. 2, pp. 624-634, June 20219.

[17] L. Zang, X. Zhang and B. Guo, "Federated Deep Reinforcement Learning for Online Task Offloading and Resource Allocation in WPC-MEC Networks," in IEEE Access, vol. 10, pp. 9856-9867, 2022.

[18] M. Z. Alam and A. Jamalipour, "Multi-Agent DRL-Based Hungarian Algorithm (MADRLHA) for Task Offloading in Multi-Access Edge Computing Internet of Vehicles (IoVs)," in IEEE Transactions on Wireless Communications, vol. 21, no. 9, pp. 7641-7652, Sept. 2022.

[19] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu and X. Shen, ”Energy Efficient Dynamic Offloading in Mobile Edge Computing for Internet of Things,” in IEEE Transactions on Cloud Computing, vol. 9, no. 3, pp. 1050-1060, 1 July-Sept. 2021.

[20] J. Liu et al., ”Reliability-Enhanced Task Offloading in Mobile Edge Computing Environments,” in IEEE Internet of Things Journal, vol. 9, no. 13, pp. 10382-10396, 1 July1, 2022.

[21] H. Guo and J. Liu, ”Collaborative Computation Offloading for Multiaccess Edge Computing Over Fiber?CWireless Networks,” in IEEE Transactions on Vehicular Technology, vol. 67, no. 5, pp. 4514-4526, May 2018.

[22] J. Ren, G. Yu, Y. Cai and Y. He, ”Latency Optimization for Resource Allocation in Mobile-Edge Computation Offloading,” in IEEE Transactions on Wireless Communications, vol. 17, no. 8, pp. 5506-5519, Aug. 2018.

[23] Wang, Yuting, Xiaofan Han, and Shunfu Jin. ”MAP based modeling method and performance study of a task offloading scheme with time-correlated traffic and VM repair in MEC systems.” Wireless Networks 29.1 (2023): 47-68.

[24] W. Fan, J. Liu, M. Hua, F. Wu and Y. Liu, ”Joint Task Offloading and Resource Allocation for Multi-Access Edge Computing Assisted by Parked and Moving Vehicles,” in IEEE Transactions on Vehicular Technology, vol. 71, no. 5, pp. 5314-5330, May 2022.

[25] Yang, Shicheng, Gongwei Lee, and Liang Huang. ”Deep learning-based dynamic computation task offloading for mobile edge computing networks.” Sensors 22.11 (2022): 4088.

[26] J. Ren, G. Yu, Y. He and G. Y. Li, ”Collaborative Cloud and Edge Computing for Latency Minimization,” in IEEE Transactions on Vehicular Technology, vol. 68, no. 5, pp. 5031-5044, May 2019.

[27] Y. Dai, D. Xu, S. Maharjan and Y. Zhang, ”Joint Computation Offloading and User Association in Multi-Task Mobile Edge Computing,” in IEEE Transactions on Vehicular Technology, vol. 67, no. 12, pp. 12313-12325, Dec. 2018.

[28] D. Gupta, A. Moudgil, S. Wadhwa and V. Solanki, ”Efficient Data Caching and Computation Offloading Strategy for Edge Network,” 2022 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 2022, pp. 1-5.

[29] Y. Hao, M. Chen, L. Hu, M. S. Hossain and A. Ghoneim, ”Energy Efficient Task Caching and Offloading for Mobile Edge Computing,” in IEEE Access, vol. 6, pp. 11365-11373, 2018.

[30] Q. Shen, B. -J. Hu and E. Xia, "Dependency-Aware Task Offloading and Service Caching in Vehicular Edge Computing," in IEEE Transactions on Vehicular Technology, vol. 71, no. 12, pp. 13182-13197, Dec. 2022.

[31] J. Chen, H. Wu, P. Yang, F. Lyu and X. Shen, "Cooperative Edge Caching With Location-Based and Popular Contents for Vehicular Networks," in IEEE Transactions on Vehicular Technology, vol. 69, no. 9, pp. 10291-10305, Sept. 2020.

[32] B. Gao, Z. Zhou, F. Liu, F. Xu and B. Li, "An Online Framework for Joint Network Selection and Service Placement in Mobile Edge Computing," in IEEE Transactions on Mobile Computing, vol. 21, no. 11, pp. 3836-3851, 1 Nov. 2022.

[33] T. Wang, Y. Zhang, N. N. Xiong, S. Wan, S. Shen and S. Huang, "An Effective Edge-Intelligent Service Placement Technology for 5G-and-Beyond Industrial IoT," in IEEE Transactions on Industrial Informatics, vol. 18, no. 6, pp. 4148-4157, June 2022.

[34] X. Zhang, Z. Li, C. Lai and J. Zhang, "Joint Edge Server Placement and Service Placement in Mobile-Edge Computing," in IEEE Internet of Things Journal, vol. 9, no. 13, pp. 11261-11274, 1 July1, 2022.

[35] Q. Ren, O. Abbasi, G. K. Kurt, H. Yanikomeroglu and J. Chen, "Caching and Computation Offloading in High Altitude Platform Station (HAPS) Assisted Intelligent Transportation Systems," in IEEE Transactions on Wireless Communications, vol. 21, no. 11, pp. 9010-9024, Nov. 2022.

[36] Z. Ning et al., "Intelligent Edge Computing in Internet of Vehicles: A Joint Computation Offloading and Caching Solution," in IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 4, pp. 2212-2225, April 2021.

[37] F. Zeng, K. Zhang, L. Wu and J. Wu, "Efficient Caching in Vehicular Edge Computing Based on Edge-Cloud Collaboration," in IEEE Transactions on Vehicular Technology, vol. 72, no. 2, pp. 2468-2481, Feb. 2023.

[38] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek and H. Jin, "Online Collaborative Data Caching in Edge Computing," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 2, pp. 281-294, 1 Feb. 2021.

[39] G. Zheng, C. Xu, H. Long and Y. Sheng, "Service Caching Based Task Offloading and Resource Allocation in Multi-UAV Assisted MEC Networks," 2021 IEEE/CIC International Conference on Communications in China (ICCC), Xiamen, China, 2021, pp. 1024-1029.

[40] C. Tang, C. Zhu, H. Wu, Q. Li and J. J. P. C. Rodrigues, "Toward Response Time Minimization Considering Energy Consumption in Caching-Assisted Vehicular Edge Computing," in IEEE Internet of Things Journal, vol. 9, no. 7, pp. 5051-5064, 1 April1, 2022.

[41] M. Guo, W. Wang, X. Huang, Y. Chen, L. Zhang and L. Chen, "Lyapunov-Based Partial Computation Offloading for Multiple Mobile Devices Enabled by Harvested Energy in MEC," in IEEE Internet of Things Journal, vol. 9, no. 11, pp. 9025-9035, 1 June1, 2022.

[42] W. Chen, D. Wang and K. Li, "Multi-User Multi-Task Computation Offloading in Green Mobile Edge Cloud Computing," in IEEE Transactions on Services Computing, vol. 12, no. 5, pp. 726-738, 1 Sept.-Oct. 2019.

[43] J. Zhao, Q. Li, Y. Gong and K. Zhang, "Computation Offloading and Resource Allocation For Cloud Assisted Mobile Edge Computing in Vehicular Networks," in IEEE Transactions on Vehicular Technology, vol. 68, no. 8, pp. 7944-7956, Aug. 2019.

[44] D. Chen et al., "Matching-Theory-Based Low-Latency Scheme for Multitask Federated Learning in MEC Networks," in IEEE Internet of Things Journal, vol. 8, no. 14, pp. 11415-11426, 15 July15, 2021.

[45] H. Wu et al., "Delay-Minimized Edge Caching in Heterogeneous Vehicular Networks: A Matching-Based Approach," in IEEE Transactions on Wireless Communications, vol. 19, no. 10, pp. 6409-6424, Oct. 2020.

[46] H. Feng, S. Guo, L. Yang and Y. Yang, "Collaborative Data Caching and Computation Offloading for Multi-Service Mobile Edge Computing," in IEEE Transactions on Vehicular Technology, vol. 70, no. 9, pp. 9408-9422, Sept. 2021.

[47] J. Xu, L. Chen and P. Zhou, "Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks," IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, Honolulu, HI, USA, 2018, pp. 207-215.

[48] J. Zhao, X. Sun, Q. Li and X. Ma, "Edge Caching and Computation Management for Real-Time Internet of Vehicles: An Online and Distributed Approach," in IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 4, pp. 2183-2197, April 2021.