Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

A cloud–edge–device collaborative offloading scheme with heterogeneous tasks and its performance evaluation*

Xiaojun BAI[1,2], Yang ZHANG[1,2], Haixing WU[1,2], Yuting WANG[1,2], Shunfu JIN†‡[1,2]

[1]*School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China*

[2]*Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Yanshan University, Qinhuangdao 066004, China*

†E-mail: jsf@ysu.edu.cn

Received Feb. 28, 2023; Revision accepted Aug. 6, 2023; Crosschecked Oct. 26, 2023; Published online Dec. 1, 2023

Abstract: How to collaboratively offload tasks between user devices, edge networks (ENs), and cloud data centers is an interesting and challenging research topic. In this paper, we investigate the offloading decision, analytical modeling, and system parameter optimization problem in a collaborative cloud–edge–device environment, aiming to trade off different performance measures. According to the differentiated delay requirements of tasks, we classify the tasks into delay-sensitive and delay-tolerant tasks. To meet the delay requirements of delay-sensitive tasks and process as many delay-tolerant tasks as possible, we propose a cloud–edge–device collaborative task offloading scheme, in which delay-sensitive and delay-tolerant tasks follow the access threshold policy and the loss policy, respectively. We establish a four-dimensional continuous-time Markov chain as the system model. By using the Gauss–Seidel method, we derive the stationary probability distribution of the system model. Accordingly, we present the blocking rate of delay-sensitive tasks and the average delay of these two types of tasks. Numerical experiments are conducted and analyzed to evaluate the system performance, and numerical simulations are presented to evaluate and validate the effectiveness of the proposed task offloading scheme. Finally, we optimize the access threshold in the EN buffer to obtain the minimum system cost with different proportions of delay-sensitive tasks.

Key words: Edge computing; Offloading scheme; Cloud–edge–device collaboration; Markov chain; Cost function
https://doi.org/10.1631/FITEE.2300128 **CLC number:** TP393

1 Introduction

Recent advances in the Internet of Things (IoT) have facilitated the explosive growth of smart connected devices and the rapid development of emerging IoT applications, such as smart city, smart vehicles, and smart healthcare, which require ever greater computational power and higher quality of service (QoS) (Mao et al., 2017; Stoyanova et al., 2020).

Traditional cloud computing is a centralized computing paradigm. Although the cloud computing paradigm offers many benefits, the classical cloud computing framework falls short in meeting some QoS requirements of numerous emerging IoT applications to a large extent. These requirements include ultra-low delay, location awareness, and privacy concerns using public cloud platforms (Muniswamaiah et al., 2021; Wang YZ et al., 2023). In addition, based on the inherently centralized architecture of the cloud computing paradigm, transmitting a huge amount of data from the IoT devices to the cloud data centers (CDCs) would easily cause congestion in wide area networks (WANs), thus resulting in additional delay. Furthermore, due to a variety of factors, including the need to process and store a large amount of generated data and the complexity

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 62273292, 62276226, and 61973261)

ORCID: Shunfu JIN, https://orcid.org/0000-0002-5845-5601

associated with transmitting a large amount of data through WANs with limited bandwidth, the unprecedented growth of connected IoT devices continues to put great pressure on the cloud.

Edge computing proposed as a new computing paradigm extends computational resources to the edge of the networks and enables data to be processed and analyzed locally in proximity to end-users (Akhlaqi and Hanapi, 2023). With the ubiquitous and intelligent development of the Internet of Everything (IoE), the cloud–edge–device collaboration architecture, which integrates the computing power of the cloud layer, the edge layer, and the terminal layer, will play an important role. The integrated cloud–edge–device collaboration architecture is expected largely to meet the QoS demands of delay-sensitive IoT applications and to reduce network bandwidth usage, while making full use of the benefits of public cloud platforms for the computationally intensive processing and storage requirements.

To deliver high-performance and ultra-low-delay solutions, the edge computing paradigm itself brings a new set of challenges (Vhora and Gandhi, 2020). One of the fundamental challenges in the cloud–edge–device collaboration architecture is the question of when and where to offload tasks from user devices (UDs). UDs can choose a nearby edge server or the remote cloud to offload tasks, which could be based on considerations such as delay and the amounts of computation and resources required. As a result, the dynamic scheduling of tasks between power-constrained UDs, resource-constrained edge nodes, and remote cloud nodes is a combination optimization problem and should be solved efficiently to fully exploit the power of this novel computing paradigm.

Furthermore, different IoT applications in the UDs tend to generate diverse computation tasks, and the arrivals of computation tasks are often random and unpredictable (Xia et al., 2020). Therefore, the problem of treating multiple types of tasks with different QoS requirements in edge computing is complex when making offloading decisions, which is also a great challenge that should be efficiently addressed. To distinguish task types, the requests generated by UDs are usually classified into delay-sensitive tasks and delay-tolerant tasks based on time constraints (Ma et al., 2021; Djigal et al., 2022; Ai et al., 2023). Delay-sensitive tasks have strict timing constraints and are required to be completed within specific time frames or by particular deadlines. For example, in delay-sensitive and real-time applications, such as autonomous vehicles, online gaming, virtual reality, and mobile augmented reality, tasks related to monitoring, decision-making, and control need to be completed within strict time limits to ensure effectiveness and user experience. Delay-tolerant tasks can tolerate some degree of delay in their execution or data delivery without significant negative impacts. For example, in delay-tolerant and non-real-time applications, such as email clients, file synchronization, ride-sharing services, and environmental monitoring, tasks can be postponed or completed during off-peak times without causing significant disruptions or negative consequences.

The division of tasks in cloud and edge computing helps prioritize and allocate resources effectively. It ensures that delay-sensitive tasks receive appropriate attention and resources to meet strict time constraints, while allowing flexibility in completing delay-tolerant tasks that can tolerate delay without significant consequences. This is similar to the simple triage process of sorting patients in sequence of who needs the most help in the emergency room. Generally speaking, delay-sensitive tasks tend to obtain timely services on the local UDs or on the edge networks (ENs) to avoid high queuing delay. The delay-tolerant tasks tend to obtain services on the CDC to improve the system throughput.

However, the offloading decision of each task is related to not only the task type but also the traffic load. It is reasonable to formulate personalized and dynamic task offloading schemes in the edge computing environments. In addition, since the same system configurations may have inconsistent effects on different system performance measures, this requires identification of the optimal system configurations (by constructing a multi-objective optimization problem) when investigating trade-off among different system performance measures.

To address the above challenges, this research is devoted to the characterization of the scheduling process of diverse tasks and the trade-off of different performance measures. In this paper, we propose a task offloading scheme with task classification in the cloud–edge–device collaboration architecture. To meet the differentiated requirements of delay-sensitive and delay-tolerant tasks and to

balance the task loads on the UDs, ENs, and CDCs, we establish a comprehensive system model with two types of computation tasks and homogeneous edge servers. Different from a vast majority of task offloading strategies (Li YZ et al., 2020; Guo XB et al., 2022; He XQ et al., 2022) that handle homogeneous tasks with different data sizes, we design the computational offloading scheme from the perspective of task classification. We evaluate the long-term performance of the system with our proposed scheme and optimize the system performance.

The main contributions of this work are summarized as follows:

1. Learning from the triage mechanism most often used in medicine, we propose a task offloading scheme with heterogeneous tasks in a stable cloud–edge–device collaborative environment. Considering admission control, the delay-sensitive tasks are allocated to the UDs or ENs following an access threshold policy; the delay-tolerant tasks are allocated to the UDs or ENs following the loss policy.

2. Based on the proposed task offloading scheme, we consider the cloud–edge–device collaboration architecture composed of three queuing sub-models on the local UDs, ENs, and CDCs, separately. The virtual machines (VMs) on the ENs and CDCs are abstracted as the servers, the tasks are abstracted as the customers, and the admission control mechanism with threshold is abstracted as the N-policy.

3. By combining the number of delay-tolerant tasks on the ENs, the number of tasks (including delay-sensitive and delay-tolerant tasks) on the ENs, the number of delay-tolerant tasks on the UDs, and the number of tasks (including delay-sensitive and delay-tolerant tasks) on the UDs, we establish a four-dimensional continuous-time Markov chain (CTMC) as the system model. From a steady-state perspective, by constructing the one-step state transition matrix and using the Gauss–Seidel method, we derive the stationary probability distribution of the system model, and evaluate the system's long-term performance on the proposed task offloading scheme.

4. To trade off different performance measures, we build a system cost function by assigning weights to different objectives. We investigate the optimal access threshold in the EN buffer to obtain the minimum system cost with different proportions of delay-sensitive tasks.

2 Related works

At present, most of existing works on edge computing focus on resource deployment (Tong et al., 2020; Chahoud et al., 2023) and task offloading (Islam et al., 2021; Saeik et al., 2021; Jayanetti et al., 2022; Liao et al., 2022). Task offloading strategies and modeling methods play a critical role in evaluating the performance of the edge computing system (Zheng et al., 2020). In this section, related works on task offloading scheme in an edge computing environment are reviewed and summarized from two aspects: collaborative task offloading and full and partial task offloading. In addition, the literature on the Markov chain model applied to describe the stochastic behaviors of systems is summarized.

2.1 Collaborative task offloading

With the explosive growth of mobile users and emerging applications, resource-limited UDs and ENs are facing the overload problem (Hossain et al., 2020). To efficiently process multiple types of task data, improve resource utilization, and achieve better user experience, collaboration between cloud computing and edge computing has become an important development trend in the IoT era (Feng et al., 2022).

Zhang JY et al. (2022) proposed an end-to-end (E2E) fine-grained service offloading mechanism in collaborative edge computing based on a network-adaptive service graph reconstruction algorithm, which can effectively balance network resource utilization and reduce the average service delay. Su et al. (2023) proposed a cloud–edge collaboration framework of intelligent health-care system (IHS) combining federated learning and blockchain. Under the cloud–edge collaboration framework, a task allocation scheme was proposed to mine the value of data from IHS tasks in terms of execution delay, computing complexity, and power consumption. Hao et al. (2019) investigated an intelligent task offloading scheme (called iTaskOffloading) for a cloud–edge collaborative system, which can provide personalized task offloading. The method of iTaskOffloading contains both coarse-grained and fine-grained computing to shorten task duration. Wu et al. (2019) proposed a cloud–edge collaborative multi-task computing offloading scheme by considering both delay and energy cost. They used an adaptive particle

swarm optimization algorithm based on nonlinear exponential inertia weight to obtain the optimal offloading decision scheme. Yang et al. (2021) proposed a trusted edge computing framework (called EdgeKeeper) based on cloud–edge fusion for adapting to the ubiquitous power IoT. Zhang MJ et al. (2022) built an edge-native task scheduling system to manage the geographically distributed edge resources and facilitate efficient task scheduling in collaborative edge computing. Two online scheduling algorithms were designed to optimally decide the task-allocation, bandwidth-allocation, and flow-routing policies by jointly considering the data and the computing and networking resources. He JY et al. (2020) proposed an online auction-based incentive mechanism for a device-to-device (D2D) collaboration system with multiple types of resources. The online incentive mechanism optimizes the long-term system welfare without knowledge of future information. Thai et al. (2020) transformed the problem of joint task offloading and resource allocation in a cloud–edge computing architecture into a mixed-integer nonlinear programming problem. They developed an approximation algorithm to obtain the optimal solutions iteratively. Gholami and Baras (2021) formulated a multi-objective mixed-integer linear program to solve the problem of energy- and delay-efficient application offloading in a collaborative cloud–edge–device environment. They proposed an approximation algorithm based on linear program relaxation and rounding to address time complexity.

These studies focused mainly on the investigation of task offloading in vertical collaboration scenarios based on cloud–edge and cloud–edge–device architectures or in horizontal collaboration scenarios based on E2E and D2D architectures. However, in these studies, different collaborative offloading strategies are not considered for different types of tasks. The cloud–edge–device collaboration architecture is still in the early phase of development under the collaborative edge computing strategy. In this paper, we investigate differentiated task offloading strategies for different types of tasks in a cloud–edge–device collaboration environment.

2.2 Full and partial task offloading

In general, decisions about computation offloading in the mobile edge computing (MEC) environment can be categorized into two main types: full offloading and partial offloading. Full offloading enables UDs to fully release resources, and the offloading task is handled independently by ENs or CDCs. Partial offloading in edge computing environments can significantly reduce the average service delay by processing computation tasks on UDs, ENs, and CDCs in parallel. If the application is indivisible or the parts after separation are closely related, the policy of partial offloading decision cannot be adopted. Extensive research has been conducted to study full and partial task offloading, with more attention being paid to the heterogeneity of edge resources, the mobility of UDs, and the multi-objective optimization of task scheduling.

Guo M et al. (2022) investigated the partial computation offloading schemes for multiple mobile devices enabled by the harvested energy in MEC environments. They formulated a nonconvex optimization problem by minimizing the energy consumption of all the mobile devices while satisfying the constraint of time delay. He XQ et al. (2022) proposed an online auction-based incentive mechanism to solve the social welfare maximization problem for full computation offloading in edge computing networks with heterogeneous edge servers. By applying the primal-dual optimization framework, they proved that the mechanism produces a good competitive ratio in social welfare. Tong et al. (2020) proposed an adaptive task offloading and resource allocation algorithm in MEC by considering the mobility of user equipments between base stations. The algorithm uses the deep reinforcement learning method to determine whether the task needs to be offloaded and which edge node is selected. Zhan et al. (2020) investigated the offloading decision and resource allocation problem among multiple users. By considering the users' mobility in the process of urgent task offloading, they proposed a heuristic mobility-aware partial offloading algorithm to obtain the approximately optimal offloading scheme. Tan et al. (2022) investigated the joint optimization problem of full offloading decision, collaboration decision, and resource allocation. The problem was formulated as a mixed-integer nonlinear programming problem, which minimizes the total energy consumption of all mobile users under the constraint on computation delay. Mao et al. (2016) investigated a green MEC system with energy harvesting devices and developed an effective partial offloading

scheme in the mobile execution process. They proposed a Lyapunov optimization-based dynamic computation offloading algorithm, which jointly optimizes the offloading decision, the central processing unit (CPU) cycle frequencies, and the transmission power. Li YZ et al. (2020) proposed a deep reinforcement learning algorithm to solve the complex computation offloading problem in heterogeneous EN scenarios. Based on the real-time state of the EN and the attributes of the task, the proposed algorithm can offload different types of tasks to different sites of the EN, and more effectively use broadband and computing resources.

In the above studies on full and partial task offloading, most of the studies assume that tasks are homogeneous, while some consider the heterogeneity of tasks. With the rapid development of edge computing, tasks generated by UDs tend to be more diversified and personalized. The difference in delay sensitivity of tasks is an important factor to be considered when designing task offloading strategies in the edge computing environment. Motivated by this, in this paper, by classifying tasks into delay-sensitive and delay-tolerant tasks, we propose a full task offloading scheme with heterogeneous tasks to implement differentiated offloading decisions.

2.3 Markov chain model

Markov chains are frequently used as inherently stochastic models for modeling real-world processes. By explicitly considering the transition probabilities from one state to another, Markov chains can effectively capture the random nature of real-world systems. Markov chains provide analysis of system state transitions and steady-state distribution. Once the transition probabilities are defined, it becomes possible to study the system's behavior over an extended period. A number of studies have used the Markov chain model to capture the random behaviors of systems.

Zhao et al. (2023) proposed a task offloading scheme in an MEC system with a two-tier edge structure considering the differentiated QoS requirements from diverse tasks. They established a two-dimensional CTMC as an edge model at the edge layer and developed an improved algorithm based on the Lagrangian multiplier method to jointly optimize the offloading probabilities of different types of tasks. Bai and Jin (2022) established a five-dimensional CTMC as the system model to capture the workflow of real-time tasks and non-real-time tasks in CDCs. They applied the matrix-geometric solution and the Gauss–Seidel iterative method to analyze the steady-state distribution of the system model. Finally, they obtained the Pareto optimal solutions for trading off the overall power consumption and average waiting time of real-time tasks. Zhou et al. (2019) formulated the offloading problem in multi-server MEC systems as a nonlinear combinatorial optimization problem of computation task assignment and CPU frequency scaling. By constructing a Markov chain for the optimization problem, they proposed an approximation algorithm based on the Markov approximation framework. This algorithm found the near-optimal solution by implementing the Markov chain over all feasible configurations and performing state transitions. Kim et al. (2021) modeled the operation of cells in a mobile communication network with moving users as a multi-server queuing system. The behavior of this system was described by a multi-dimensional Markov chain. By solving the stationary distribution of this Markov chain, they obtained the key characteristics of user QoS.

The above studies demonstrate that Markov chains facilitate the analysis of long-term behaviors and steady-state properties of systems and can capture the random behaviors of different types of tasks. Nevertheless, it is important to note that in the literature, Markov chains for capturing random behaviors of different types of tasks in the edge computing environment are predominantly established based on random offloading schemes. These schemes assume that the state of each layer is unobservable; i.e., the offloading probability of each layer is independent of the current local load, and this enables each layer to maintain the Markov property in a cloud–edge–device collaborative environment. Additionally, note that state-observable offloading schemes can more effectively balance system load and ensure the QoS of different types of tasks, but they compromise independent Markov properties of each layer and make the theoretical analysis more difficult.

To the best of our knowledge, the investigation of offloading schemes involving state-observable threshold control and the construction of multi-dimensional Markov chains based on such schemes have not been studied before. To fill in the gap, in this paper, based on an offloading scheme that

incorporates the access threshold policy and the loss policy, we establish a four-dimensional CTMC by simultaneously characterizing the scheduling processes of UDs, ENs, and CDCs. By applying the four-dimensional CTMC, we aim to effectively capture and model the entire process of cloud–edge–device collaborative scheduling of diverse tasks, while preserving the Markov property.

The primary characteristics and limitations of the cited literature are summarized in Table 1.

3 Task offloading scheme and system model

In this section, we propose a cloud–edge–device collaborative task offloading scheme, in which the delay-sensitive and delay-tolerant tasks obtain services on UDs and ENs following the access threshold policy and the loss policy, respectively. Then, we establish a four-dimensional Markov chain as the system model to capture the stochastic behaviors of these two types of tasks in the edge computing system with the proposed scheme.

3.1 Task offloading scheme

In general, UDs are far away from CDCs, so the transmission of tasks from the UDs to the CDCs will bring high propagation delay. To improve the experience quality of remote users, edge computing technologies are applied by the service provider.

By considering the cloud–edge–device collaboration architecture, we investigate the task offloading scheme in the edge computing system consisting of a UD, an EN, and a CDC. EN is deployed at the edge of the wireless network, between the UD and CDC. A processor and a buffer are deployed in the UD. One task can be processed on the UD at a time. The physical machine (PM) in the EN can be virtualized into several homogeneous VMs, and these VMs share a single buffer. Multiple tasks can be processed on the EN simultaneously. Enough VMs are deployed in the CDC to provide timely services for the offloaded tasks, and these VMs are homogeneous as well. By setting task schedulers in the UD and EN separately, we propose a task offloading scheme in the cloud–edge–device collaboration architecture.

Table 1 Summary of literature review

Reference	Offloading	Collaboration	Heterogeneity	Objective(s)	Shortcoming
Zhang JY et al. (2022)	Partial	Device–device		ORUBD, delay	Addressing the problem of scheduling tasks only among edge nodes
Su et al. (2023)	Full	Cloud–edge	✓	Delay, power	Slow convergence
Hao et al. (2019)	Partial	Cloud–edge		Delay	No consideration of heterogeneous resources
Wu et al. (2019)	Full	Cloud–edge		Delay, energy	No consideration of dependencies between tasks
Yang et al. (2021)	Partial	Cloud–edge	✓	Delay	High computational power required for training
Zhang MJ et al. (2022)	Full	Edge–device		Throughput, latency	No consideration of task partition and allocation
He JY et al. (2020)	Partial	Device–device	✓	Social welfare	No investigation of cloud–edge collaborative offloading
Guo M et al. (2022)	Partial	Edge–device		Energy, discarding ratio	Considering only one edge node
He XQ et al. (2022)	Full	Edge–device		Social welfare	No consideration of cloud collaboration
Tong et al. (2020)	Full	Edge–device		Delay, energy	No consideration of cloud collaboration
Zhan et al. (2020)	Partial	Edge–device		Delay, energy	No consideration of collaboration among multiple edge nodes
Tan et al. (2022)	Full	Device–device		Energy	No consideration of cloud–edge collaboration
Mao et al. (2016)	Partial	Edge–device		Latency, failure	No consideration of edge server virtualization
Gholami and Baras (2021)	Partial	Cloud–edge–device		Latency, energy	No consideration of dependencies between tasks
Zhao et al. (2023)	Partial	Edge–device	✓	Latency, power	No consideration of cloud collaboration
This study	Full	Cloud–edge–device	✓	Latency, blocking rate	

ORUBD: overall resource utilization balance degree

The working principle of the task offloading scheme is shown in Fig. 1.

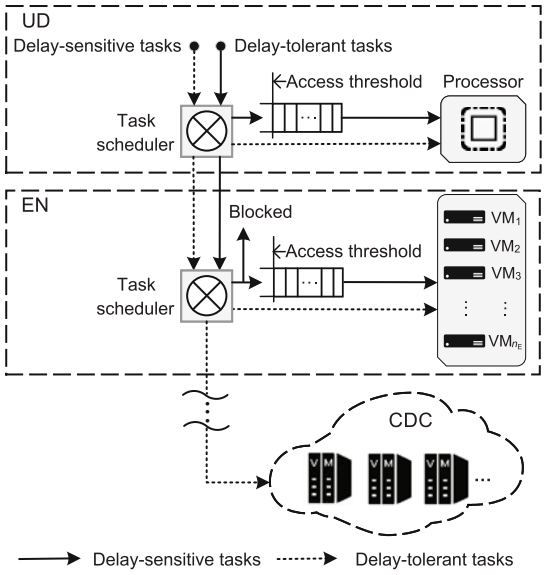

Fig. 1 Working principle of the task offloading scheme (CDC: cloud data center; EN: edge network; UD: user device)

According to the admission control policy, we set two access thresholds, H_{U} in the UD buffer and H_{E} in the EN buffer, to control the number of delay-sensitive tasks. The delay-sensitive tasks are processed on the UD or EN, and the delay-tolerant tasks can be offloaded to the CDC if necessary.

When a delay-sensitive task is generated in the UD, if the number of tasks in the UD buffer does not reach the access threshold H_{U}, the task is processed on the UD; otherwise, the task scheduler offloads the delay-sensitive task to the EN. To reduce the time occupied by delay-tolerant tasks, when a delay-tolerant task is generated, if the processor of the UD is idle, the task is processed locally; otherwise, the task is offloaded to the EN for processing.

When a delay-sensitive task arrives at the EN, if the number of tasks in the EN buffer does not reach the access threshold H_{E}, the task is processed on the EN; otherwise, the delay-sensitive task leaves the edge computing environment due to blocking. In practical applications, some of the blocked tasks may be lost, and others may be sent to other ENs to obtain services. When a delay-tolerant task arrives at the EN, if there is at least one idle VM in the EN, the task is offloaded to the EN to obtain services; otherwise, the task is further offloaded to the CDC. Due to the increased number of VMs in the CDC, the delay-tolerant tasks offloaded to the CDC are usually processed immediately without waiting. Considering the resource limitation in the UD and EN, the greater the number of tasks in the UD or EN, the longer the delay in the service of a delay-sensitive task. So, the offloading of delay-sensitive tasks follows an access threshold policy. The access threshold policy is defined by the set of integer numbers known as thresholds for granting or denying access to the system resources. When the number of customers in the queue exceeds a predetermined threshold, the newly arriving customers are not permitted to join the queue, while new customers are again allowed to gain access to the system resources and join the queue once the number of customers in the queue drops below the corresponding threshold. When a delay-sensitive task is generated, if the number of delay-sensitive tasks in the UD buffer does not reach the access threshold H_{U}, the newly generated delay-sensitive task will be processed on the UD; otherwise, it will be offloaded to the EN by the UD scheduler. When a delay-sensitive task is offloaded to the EN, if the number of delay-sensitive tasks in the EN buffer does not reach the access threshold H_{E}, the newly offloaded delay-sensitive task will be processed on the EN; otherwise, it will be blocked. In this way, the delay-sensitive tasks can avoid high latency.

The workflow of delay-sensitive tasks under the proposed scheme is demonstrated in Fig. 2.

The offloading of delay-tolerant tasks follows a special loss policy. There is no queue in a system with the loss policy. When a new customer arrives, the system immediately assesses its resources. If the system has sufficient resources to serve the new customer, the customer will be admitted immediately and provided the required services. However, if the system has reached its processing limit, the newly arriving customer is lost to find other systems. Based on the characteristics of the loss policy, we assume that delay-tolerant tasks do not enter the UD buffer or the EN buffer. When a delay-tolerant task is generated, if the UD processor is idle, the newly generated delay-tolerant task will be processed on the UD; otherwise, the UD scheduler will offload the newly generated delay-tolerant task to the EN. Similarly, when a delay-tolerant task is offloaded to the EN, if there is at least one idle VM in the EN, the

newly offloaded delay-tolerant task will be processed on the EN; otherwise, the EN scheduler will offload the newly arriving delay-tolerant task to the CDC. When a delay-tolerant task is offloaded to the CDC after long-distance transmission, the delay-tolerant task can be processed on one of the cloud VMs.

The workflow of delay-tolerant tasks under the proposed scheme is demonstrated in Fig. 3.

Under the proposed task offloading scheme, the requirements of these two types of tasks are taken into account. When the traffic load of the edge computing system is low, both delay-sensitive tasks and delay-tolerant tasks have an opportunity to be processed on the UD or EN. When the traffic load is high, the UD and EN are occupied mainly by delay-sensitive tasks, and delay-tolerant tasks are usually offloaded to the CDC to obtain services. When the load of delay-sensitive tasks in the system is high, the response performance of delay-sensitive tasks is guaranteed at the cost of a certain blocking rate.

3.2 System model

In this subsection, we present the system model studied in this paper, i.e., a four-dimensional CTMC based on the proposed task offloading scheme in Section 3.1. For ease of reference, we list the key notations used in our system model in Table 2.

Task arrivals are supposed to be independent of each other. Considering the continuous-time structure, we assume that the arrivals of delay-sensitive

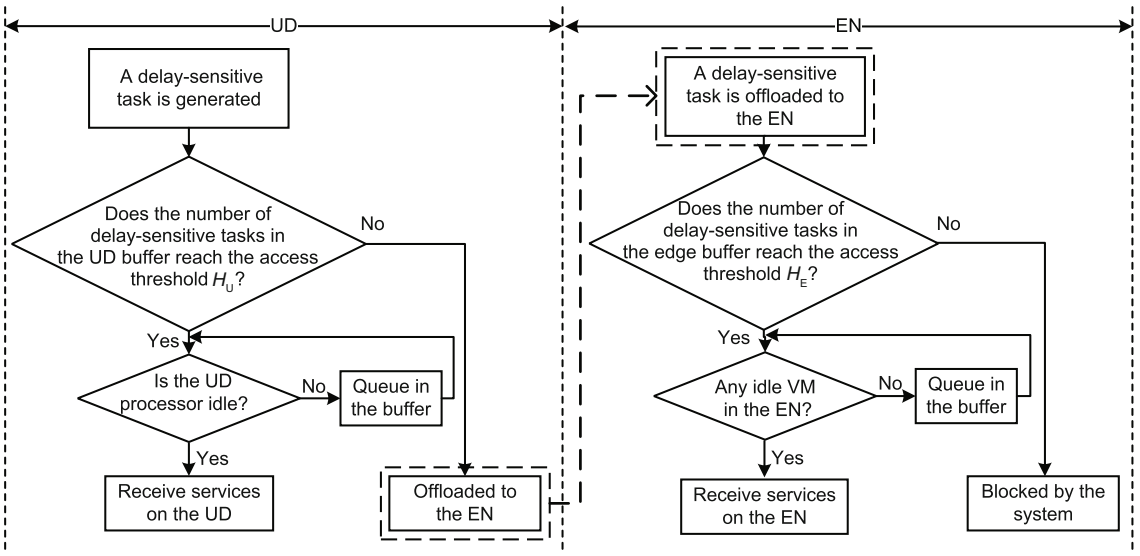

Fig. 2 Workflow of delay-sensitive tasks under the proposed scheme (EN: edge network; UD: user device; VM: virtual machine)

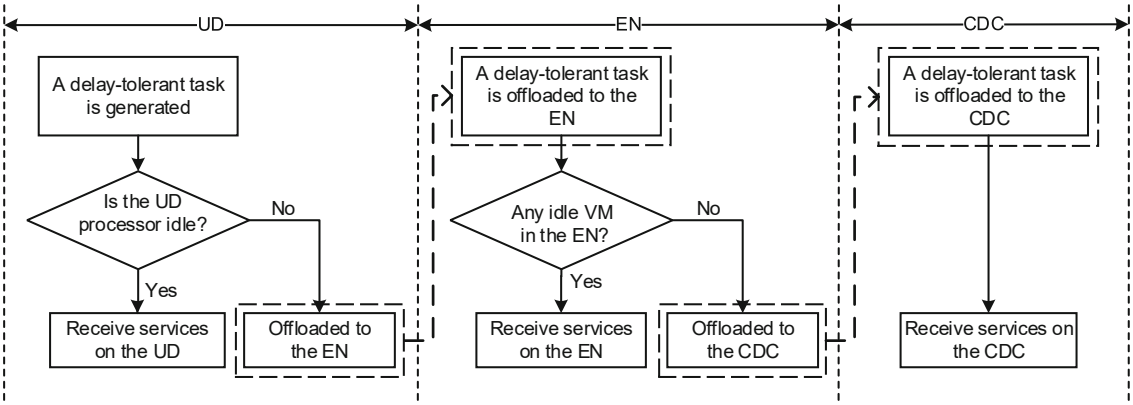

Fig. 3 Workflow of delay-tolerant tasks under the proposed scheme (CDC: cloud data center; EN: edge network; UD: user device; VM: virtual machine)

Table 2 Notations used in the system model

Notation	Description
λ_1	Arrival rate of delay-sensitive tasks
λ_2	Arrival rate of delay-tolerant tasks
μ_{11}	Local service rate of delay-sensitive tasks
μ_{21}	Local service rate of delay-tolerant tasks
μ_{12}	Edge service rate of delay-sensitive tasks
μ_{22}	Edge service rate of delay-tolerant tasks
μ_{23}	Cloud service rate of delay-tolerant tasks
n_{E}	Number of edge VMs in the EN
H_{U}	Access threshold in the UD buffer
H_{E}	Access threshold in the EN buffer

VM: virtual machine; EN: edge network; UD: user device

and delay-tolerant tasks follow the Poisson process with parameters λ_1 ($\lambda_1 \geq 0$) and λ_2 ($\lambda_2 \geq 0$), respectively. We call λ_1 and λ_2 the arrival rates of delay-sensitive and delay-tolerant tasks, respectively.

We note that each task has a different service time, which can be assumed as a random variable. We assume that the service time of the delay-sensitive tasks processed on the UD and that of the delay-sensitive tasks processed on the EN follow the exponential distribution with parameters μ_{11} and μ_{12} ($\mu_{12} \geq \mu_{11} > 0$), respectively. We call μ_{11} and μ_{12} the local service rate and the edge service rate of delay-sensitive tasks, respectively. We assume that the service time of the delay-tolerant tasks processed on the UD and that of the delay-tolerant tasks processed on the EN follow the exponential distribution with parameters μ_{21} ($\mu_{21} \geq 0$) and μ_{22} ($\mu_{22} \geq \mu_{21}$), respectively. We also assume that the service time of the delay-tolerant tasks processed on the CDC follows the exponential distribution with parameter μ_{23} ($\mu_{23} \geq \mu_{22}$). We call μ_{21}, μ_{22}, and μ_{23} the local service rate, edge service rate, and cloud service rate of delay-tolerant tasks, respectively. Without loss of generality, we assume that the service rates for these two types of tasks are different, namely, $\mu_{11} \neq \mu_{21}, \mu_{12} \neq \mu_{22}$. The service time of tasks is supposed to be independent of each other. Tasks in the UD and EN are processed following the first-come-first-served (FCFS) discipline.

From the workflows of delay-sensitive and delay-tolerant tasks in Figs. 2 and 3, respectively, we find that only the delay-sensitive tasks generated when the number of tasks in the UD buffer reaches the access threshold H_E and the delay-tolerant tasks generated when the UD is busy can be offloaded to the EN. That is to say, the arrivals of these two types of tasks at the EN are no longer subject to the Poisson process. For this, we record the number of delay-tolerant tasks in the EN, the number of these two types of tasks in the EN, the number of delay-tolerant tasks in the UD, and the number of these two types of tasks in the UD to construct a four-dimensional Markov chain.

To keep track of the behavior of the system, we introduce the following notations:

$X(t)$: the number of delay-tolerant tasks in the EN at instant t, commonly referred to as the system level. $X(t) = i,\ i \in \{0, 1, \ldots, n_{\mathrm{E}}\}$.

$Y(t)$: the number of tasks (including delay-sensitive and delay-tolerant tasks) in the EN at instant t, commonly referred to as the system state. $Y(t) = j,\ j \in \{i, i+1, \ldots, n_{\mathrm{E}} + H_{\mathrm{E}}\}$.

$Z(t)$: the number of delay-tolerant tasks in the UD at instant t. $Z(t) = k,\ k \in \{0, 1\}$.

$G(t)$: the number of tasks (including delay-sensitive and delay-tolerant tasks) in the UD at instant t. $G(t) = r,\ r \in \{k, k+1, \ldots, 1 + H_{\mathrm{U}}\}$.

We establish a four-dimensional CTMC $\{(X(t), Y(t), Z(t), G(t)), t \geq 0\}$ as the system model to describe the proposed task offloading scheme and further optimize the system performance. The state space of this Markov chain is given as follows:

$$\begin{aligned}\Omega = \Big\{ & (i,j,k,r) \mid i \in \{0,1,\ldots,n_{\mathrm{E}}\}, \\ & j \in \{i, i+1, \ldots, n_{\mathrm{E}} + H_{\mathrm{E}}\}, \\ & k \in \{0,1\}, r \in \{k, k+1, \ldots, 1+H_{\mathrm{U}}\} \Big\}.\end{aligned} \tag{1}$$

We define $\pi_{i,j,k,r}$ as the stationary probability of the four-dimensional CTMC when the number of delay-tolerant tasks in the EN is i, the number of tasks in the EN is j, the number of delay-tolerant tasks in the UD is k, and the number of tasks in the UD is r. The term $\pi_{i,j,k,r}$ is then given as follows:

$$\begin{aligned}\pi_{i,j,k,r} = & \lim_{t \to \infty} P\{X(t) = i, Y(t) = j, Z(t) = k, G(t) = r\}, \\ & (i,j,k,r) \in \Omega.\end{aligned} \tag{2}$$

We define vector $\boldsymbol{\pi}_i$ as the stationary probability vector when the number of delay-tolerant tasks in the EN is i. Then, the stationary probability distribution $\boldsymbol{\Pi}$ of the four-dimensional CTMC is composed of $\boldsymbol{\pi}_i, i \in \{0, 1, \ldots, n_{\mathrm{E}}\}$. $\boldsymbol{\Pi}$ is partitioned as follows:

$$\boldsymbol{\Pi} = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_{n_{\mathrm{E}}}). \tag{3}$$

4 Model analysis and performance measures

In this section, we present the generator of the four-dimensional CTMC and derive the stationary probability distribution of the system model. Then, we derive the system performance measures.

4.1 Model analysis

Let $\boldsymbol{Q}$ be the generator of the four-dimensional CTMC $\{(X(t),Y(t),Z(t),G(t)),t\geq 0\}$. Let $\boldsymbol{Q}_{u,v}$ be the one-step state transition rate matrix for the system level changing from u ($u\in\{0,1,\ldots,n_{\mathrm{E}}\}$) to v ($v\in\{0,1,\ldots,n_{\mathrm{E}}\}$). The generator $\boldsymbol{Q}$ can be given in an $(n_{\mathrm{E}}+1)\times(n_{\mathrm{E}}+1)$-block-structure form as follows:

$$\boldsymbol{Q}=\begin{bmatrix}\boldsymbol{Q}_{0,0} & \boldsymbol{Q}_{0,1} & & & & \\ \boldsymbol{Q}_{1,0} & \boldsymbol{Q}_{1,1} & \boldsymbol{Q}_{1,2} & & & \\ & \ddots & \ddots & \ddots & & \\ & & \boldsymbol{Q}_{n_{\mathrm{E}}-1,n_{\mathrm{E}}-2} & \boldsymbol{Q}_{n_{\mathrm{E}}-1,n_{\mathrm{E}}-1} & \boldsymbol{Q}_{n_{\mathrm{E}}-1,n_{\mathrm{E}}} & \\ & & & \boldsymbol{Q}_{n_{\mathrm{E}},n_{\mathrm{E}}-1} & \boldsymbol{Q}_{n_{\mathrm{E}},n_{\mathrm{E}}} & \end{bmatrix}.$$

To analyze the non-zero sub-blocks of $\boldsymbol{Q}$, we introduce notations $\boldsymbol{I}_{2H_{\mathrm{U}}+3}$ and $\boldsymbol{0}$ to denote the identity matrix with $2H_{\mathrm{U}}+3$ dimensions and the zero matrix with an appropriate dimension, respectively. The non-zero one-step state transition rate matrix $\boldsymbol{Q}_{u,v}$ of the Markov chain $\{(X(t),Y(t),Z(t),G(t)),t\geq 0\}$ can be discussed according to the changes of system levels as follows:

1. The one-step state transition rate matrix $\boldsymbol{Q}_{i,i-1}$ indicates that the system level i changes to $i-1$ via a one-step state transition, where $i\in\{1,2,\ldots,n_{\mathrm{E}}\}$. Let $N_{\mathrm{E}}=n_{\mathrm{E}}+H_{\mathrm{E}}$. $\boldsymbol{Q}_{i,i-1}$ can be given in an $(N_{\mathrm{E}}+1-i)\times(N_{\mathrm{E}}+2-i)$-block-structure form as follows:

$$\boldsymbol{Q}_{i,i-1}=\begin{bmatrix}\boldsymbol{S}_{i,i-1} & \boldsymbol{0} & & & \\ & \boldsymbol{S}_{i+1,i} & \boldsymbol{0} & & \\ & & \ddots & \ddots & \\ & & & \boldsymbol{S}_{N_{\mathrm{E}}-1,N_{\mathrm{E}}-2} & \boldsymbol{0} \\ & & & & \boldsymbol{S}_{N_{\mathrm{E}},N_{\mathrm{E}}-1}\ \boldsymbol{0}\end{bmatrix},$$

where $\boldsymbol{S}_{j,j-1}$ ($j\in\{i,i+1,\ldots,N_{\mathrm{E}}\}$) represents the transition rate sub-matrix when the system state changes from j to $j-1$. For this case, a delay-tolerant task departs from the EN; then, the number of delay-tolerant tasks in the EN changes from i to $i-1$, and the total number of tasks in the EN changes from j to $j-1$. Here, $\boldsymbol{S}_{j,j-1}$ is of order $2H_{\mathrm{U}}+3$ and is given as follows:

$$\boldsymbol{S}_{j,j-1}=i\mu_{22}\boldsymbol{I}_{2H_{\mathrm{U}}+3}. \tag{4}$$

2. The one-step state transition rate matrix $\boldsymbol{Q}_{i,i+1}$ indicates that the system level i changes to $i+1$ via a one-step state transition, where $i\in\{0,1,\ldots,n_{\mathrm{E}}-1\}$. Here, $\boldsymbol{Q}_{i,i+1}$ can be given in an $(N_{\mathrm{E}}+1-i)\times(N_{\mathrm{E}}-i)$-block-structure form as follows:

$$\boldsymbol{Q}_{i,i+1}=\begin{bmatrix}\boldsymbol{T}_{i,i+1} & & & & \\ & \boldsymbol{T}_{n_{\mathrm{E}}-1,n_{\mathrm{E}}} & & & \\ & & \boldsymbol{0} & & \\ & & & \ddots & \\ & & & & \boldsymbol{0} \\ & & & & \boldsymbol{0}\end{bmatrix},$$

where $\boldsymbol{T}_{j,j+1}$ ($j\in\{i,i+1,\ldots,N_{\mathrm{E}}-1\}$) represents the transition rate sub-matrix when the system state changes from j to $j+1$. For this case, a delay-tolerant task arrives at the EN; then, the number of delay-tolerant tasks in the EN changes from i to $i+1$, and the total number of tasks in the EN changes from j to $j+1$. Here, $\boldsymbol{T}_{j,j+1}$ is of order $2H_{\mathrm{U}}+3$ and is given as follows:

$$\boldsymbol{T}_{j,j+1}=\mathrm{diag}(0,\underbrace{\lambda_2,\lambda_2,\ldots,\lambda_2}_{2H_{\mathrm{U}}+2}). \tag{5}$$

3. The one-step state transition rate matrix $\boldsymbol{Q}_{i,i}$ indicates that system level i is fixed via a one-step state transition, where $i\in\{0,1,\ldots,n_{\mathrm{E}}\}$. Here, $\boldsymbol{Q}_{i,i}$ can be given in an $(N_{\mathrm{E}}+1-i)\times(N_{\mathrm{E}}+1-i)$-block-structure form as follows:

$$\boldsymbol{Q}_{i,i}=\begin{bmatrix}\boldsymbol{L}_{i,i} & \boldsymbol{L}_{i,i+1} & & & \\ \boldsymbol{L}_{i+1,i} & \boldsymbol{L}_{i+1,i+1} & \boldsymbol{L}_{i+1,i+2} & & \\ & \ddots & \ddots & \ddots & \\ & & \boldsymbol{L}_{d-1,d-2} & \boldsymbol{L}_{d-1,d-1} & \boldsymbol{L}_{d-1,d} \\ & & & \boldsymbol{L}_{d,d-1} & \boldsymbol{L}_{d,d}\end{bmatrix},$$

where $d=N_{\mathrm{E}}$.

In the one-step state transition rate matrix $\boldsymbol{Q}_{i,i}$, sub-matrix $\boldsymbol{L}_{j,j'}$ ($j\in\{i,i+1,\ldots,N_{\mathrm{E}}\}, j'\in\{j-1,j,j+1\}$) represents that the system state changes from j to j' for a fixed system level i. Sub-matrix $\boldsymbol{L}_{j,j'}$ is square and is of order $2H_{\mathrm{U}}+3$. According to the changes of system states, $\boldsymbol{L}_{j,j'}$ can be discussed in three cases.

Case 1: System state j changes to $j-1$ via a one-step state transition, where $j\in\{i+1,i+2,\ldots,N_{\mathrm{E}}\}$. For this case, a delay-sensitive task departs from the EN, the number of delay-tolerant tasks in the EN remains fixed, and the total number of tasks in the EN changes from j to $j-1$. The transition rate

sub-matrix $\boldsymbol{L}_{j,j-1}$ can be given as follows:

$$\boldsymbol{L}_{j,j-1}=\begin{cases}(j-i)\mu_{12}\boldsymbol{I}_{2H_{\mathrm{U}}+3},\\ \qquad j\in\{i+1,i+2,\ldots,n_{\mathrm{E}}-1-i\},\\ n_{\mathrm{E}}\mu_{12}\boldsymbol{I}_{2H_{\mathrm{U}}+3},\\ \qquad j\in\{n_{\mathrm{E}}-i,n_{\mathrm{E}}-i+1,\ldots,N_{\mathrm{E}}\}.\end{cases}$$

Case 2: System state j changes to $j+1$ via a one-step state transition, where $j\in\{i,i+1,\ldots,N_{\mathrm{E}}-1\}$. For this case, a delay-sensitive task arrives at the EN, the number of delay-tolerant tasks in the EN remains fixed, and the total number of tasks in the EN changes from j to $j+1$. The transition rate sub-matrix $\boldsymbol{L}_{j,j+1}$ can be given as follows:

$$\boldsymbol{L}_{j,j+1}=\mathrm{diag}(\underbrace{0,0,\ldots,0}_{H_{\mathrm{U}}+1},\lambda_1,\underbrace{0,0,\ldots,0}_{H_{\mathrm{U}}},\lambda_1).$$

Case 3: System state j is fixed via a one-step state transition, where $j\in\{i,i+1,\ldots,N_{\mathrm{E}}\}$. For this case, there are neither arrivals nor departures of delay-sensitive task in the EN, and the number of delay-tolerant tasks in the EN and the total number of tasks in the EN remain fixed. The transition rate sub-matrix $\boldsymbol{L}_{j,j}$ can be given as follows:

$$\boldsymbol{L}_{j,j}=\begin{bmatrix}a_{0,0} & \lambda_1 & & & & \lambda_2 & & & \\ \mu_{11} & a_{1,1} & \lambda_1 & & & & & & \\ & \ddots & \ddots & \ddots & & & & & \\ & & \mu_{11} & a_{c,c} & \lambda_1 & & & & \\ & & & \mu_{11} & a_{c+1,c+1} & & & & \\ \mu_{21} & & & & & b_{1,1} & \lambda_1 & & \\ & \ddots & & & & & \ddots & \ddots & \\ & & \mu_{21} & & & & b_{c,c} & \lambda_1 & \\ & & & \mu_{21} & & & & b_{c+1,c+1} & \end{bmatrix},$$

where $c=H_{\mathrm{U}}$, $a_{m,m}$ ($m\in\{0,1,\ldots,H_{\mathrm{U}}+1\}$) and $b_{n,n}$ ($n\in\{1,2,\ldots,H_{\mathrm{U}}+1\}$) can be discussed according to different system states j.

When system state j satisfies $i\le j\le n_{\mathrm{E}}-1$, $a_{m,m}$ and $b_{n,n}$ can be given as follows:

$$a_{m,m}=\begin{cases}-(\lambda_1+\lambda_2+(j-i)\mu_{12}+i\mu_{22}),\\ \qquad m=0,\\ -(\lambda_1+\lambda_2+\mu_{11}+(j-i)\mu_{12}+i\mu_{22}),\\ \qquad m\in\{1,2,\ldots,H_{\mathrm{U}}+1\},\end{cases}\tag{6}$$

$$b_{n,n}=-(\lambda_1+\lambda_2+\mu_{21}+(j-i)\mu_{12}+i\mu_{22}),\quad n\in\{1,2,\ldots,H_{\mathrm{U}}+1\}.\tag{7}$$

When system state j satisfies $n_{\mathrm{E}}\le j\le N_{\mathrm{E}}-1$, $a_{m,m}$ and $b_{n,n}$ can be given as follows:

$$a_{m,m}=\begin{cases}-(\lambda_1+\lambda_2+(n_{\mathrm{E}}-i)\mu_{12}+i\mu_{22}),\\ \qquad m=0,\\ -(\lambda_1+\mu_{11}+(n_{\mathrm{E}}-i)\mu_{12}+i\mu_{22}),\\ \qquad m\in\{1,2,\ldots,H_{\mathrm{U}}+1\},\end{cases}\tag{8}$$

$$b_{n,n}=-(\lambda_1+\mu_{21}+(n_{\mathrm{E}}-i)\mu_{12}+i\mu_{22}),\quad n\in\{1,2,\ldots,H_{\mathrm{U}}+1\}.\tag{9}$$

When system state j satisfies $j=N_{\mathrm{E}}$, $a_{m,m}$ and $b_{n,n}$ can be given as follows:

$$a_{m,m}=\begin{cases}-(\lambda_1+\lambda_2+(n_{\mathrm{E}}-i)\mu_{12}+i\mu_{22}),\\ \qquad m=0,\\ -(\lambda_1+\mu_{11}+(n_{\mathrm{E}}-i)\mu_{12}+i\mu_{22}),\\ \qquad m\in\{1,2,\ldots,H_{\mathrm{U}}\},\\ -(\mu_{11}+(n_{\mathrm{E}}-i)\mu_{12}+i\mu_{22}),\\ \qquad m=H_{\mathrm{U}}+1.\end{cases}\tag{10}$$

$$b_{n,n}=\begin{cases}-(\lambda_1+\mu_{21}+(n_{\mathrm{E}}-i)\mu_{12}+i\mu_{22}),\\ \qquad n\in\{1,2,\ldots,H_{\mathrm{U}}\},\\ -(\mu_{21}+(n_{\mathrm{E}}-i)\mu_{12}+i\mu_{22}),\\ \qquad n=H_{\mathrm{U}}+1.\end{cases}\tag{11}$$

Up to now, all the sub-blocks in generator $\boldsymbol{Q}$ have been presented. The block tridiagonal structure of $\boldsymbol{Q}$ indicates that the four-dimensional CTMC $\{(X(t),Y(t),Z(t),G(t)),t\ge0\}$ is non-periodic, irreducible, and positive recurrent.

It is well known that the stationary probability distribution $\boldsymbol{\Pi}$ of the four-dimensional CTMC $\{(X(t),Y(t),Z(t),G(t)),t\ge0\}$ can be obtained by solving the following linear equation:

$$\begin{cases}\boldsymbol{\Pi}\boldsymbol{Q}=\boldsymbol{0},\\ \boldsymbol{\Pi}\boldsymbol{e}=1,\end{cases}\tag{12}$$

where $\boldsymbol{e}$ is a column vector with $\frac{1}{2}(2H_{\mathrm{E}}+n_{\mathrm{E}}+2)(n_{\mathrm{E}}+1)(2H_{\mathrm{U}}+3)$ elements and all elements of the vector are equal to 1.

By using the Gauss–Seidel method, one of the iterative methods, we calculate the stationary probability distribution $\boldsymbol{\Pi}$ of the Markov chain $\{(X(t),Y(t),Z(t),G(t)),t\ge0\}$.

4.2 Performance measures

To evaluate the long-term performance of the edge computing system with the proposed task offloading scheme, we derive the blocking rate of delay-sensitive tasks, the average delay of delay-sensitive tasks, and the average delay of delay-tolerant tasks. For the blocked delay-sensitive tasks in the edge computing system, they may be lost or sent to other ENs. When calculating the average delay of delay-sensitive tasks, the blocked tasks are not included.

1. Blocking rate of delay-sensitive tasks. The blocking rate R_{b} of delay-sensitive tasks is defined as the probability that a delay-sensitive task is blocked. When the numbers of tasks in the UD buffer and EN buffer both reach their access thresholds, a newly generated delay-sensitive task will be blocked. The blocking rate R_{b} can be given as follows:

$$R_{\mathrm{b}}=\sum_{i=0}^{n_{\mathrm{E}}}\sum_{k=0}^{1}\pi_{i,n_{\mathrm{E}}+H_{\mathrm{E}},k,H_{\mathrm{U}}+1}. \tag{13}$$

2. Average delay of delay-sensitive tasks. The delay T_{r} of a delay-sensitive task is defined as the time duration from the generation instant of a delay-sensitive task in the UD to the instant at which the delay-sensitive task finishes its service on the UD processor or on the edge VMs. From Section 3, we know that there are two possible cases for each delay-sensitive task that is successfully processed.

Case 1: processed on the UD processor. We define the local allocation rate R_{11} of delay-sensitive tasks as the probability that a delay-sensitive task is processed on the UD processor locally. R_{11} can be given as follows:

$$R_{11}=\sum_{i=0}^{n_{\mathrm{E}}}\sum_{j=i}^{n_{\mathrm{E}}+H_{\mathrm{E}}}\sum_{k=0}^{1}\sum_{r=k}^{H_{\mathrm{U}}}\pi_{i,j,k,r}. \tag{14}$$

For a delay-sensitive task processed on the UD processor, the delay includes the waiting time in the UD buffer and the service time on the UD processor. We call the delay of a delay-sensitive task processed on the UD processor the local delay of a delay-sensitive task. Following Little's law, we obtain the average local delay $E[T_{11}]$ of delay-sensitive tasks as follows:

$$E[T_{11}]=\frac{1}{\lambda_1 R_{11}}\sum_{i=0}^{n_{\mathrm{E}}}\sum_{j=i}^{n_{\mathrm{E}}+H_{\mathrm{E}}}\sum_{k=0}^{1}\sum_{r=2}^{H_{\mathrm{U}}+1}(r-1)\pi_{i,j,k,r}+\frac{1}{\mu_{11}}, \tag{15}$$

where $\lambda_1 R_{11}$ is the effective arrival rate of delay-sensitive tasks at the UD.

Case 2: processed on the edge VMs. We define the edge offloading rate R_{12} of delay-sensitive tasks as the probability that a delay-sensitive task is processed on the edge VMs. R_{12} can be given as follows:

$$R_{12}=\sum_{i=0}^{n_{\mathrm{E}}}\sum_{j=i}^{n_{\mathrm{E}}+H_{\mathrm{E}}-1}\sum_{k=0}^{1}\pi_{i,j,k,H_{\mathrm{U}}+1}. \tag{16}$$

Although the distance between the UD and the EN is small in the edge computing environment, the propagation delay t_{t} from the epoch of a delay-sensitive task departing from the UD to the epoch at which the delay-sensitive task arrives at the EN cannot be ignored. For a delay-sensitive task processed on the edge VMs, the delay includes the waiting time in the EN buffer and the service time on the edge VMs. We call the delay of a delay-sensitive task processed on the edge VMs the edge delay of a delay-sensitive task. The average edge delay $E[T_{12}]$ of delay-sensitive tasks can be given as follows:

$$E[T_{12}]=\frac{1}{\lambda_1 R_{12}}\sum_{i=0}^{n_{\mathrm{E}}}\sum_{j=n_{\mathrm{E}}+1}^{n_{\mathrm{E}}+H_{\mathrm{E}}}\sum_{k=0}^{1}\sum_{r=k}^{H_{\mathrm{U}}+1}(j-n_{\mathrm{E}})\pi_{i,j,k,r}+t_{\mathrm{t}}+\frac{1}{\mu_{12}}, \tag{17}$$

where $\lambda_1 R_{12}$ is the effective arrival rate of delay-sensitive tasks at the EN.

In the edge computing system with the proposed task offloading scheme, all delay-sensitive tasks are either successfully processed or blocked. Then, R_{11}, R_{12}, and R_{b} are subject to the following condition:

$$R_{11}+R_{12}+R_{\mathrm{b}}=1. \tag{18}$$

Combining Eqs. (15), (17), and (18), we obtain the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks as follows:

$$\begin{aligned}E[T_{\mathrm{ds}}]&=\frac{R_{11}}{R_{11}+R_{12}}E[T_{11}]+\frac{R_{12}}{R_{11}+R_{12}}E[T_{12}]\\&=\frac{R_{11}}{1-R_{\mathrm{b}}}E[T_{11}]+\frac{R_{12}}{1-R_{\mathrm{b}}}E[T_{12}].\end{aligned} \tag{19}$$

3. Average delay of delay-tolerant tasks. The delay T_{n} of a delay-tolerant task is defined as the time duration from the generation instant of a delay-tolerant task to the instant at which the delay-tolerant task finishes its service on the UD processor, on the edge VMs, or on the cloud VMs. From Section 3, we know that there are three possible cases for each delay-tolerant task that is successfully processed.

Case 1: processed on the UD processor. We define the local allocation rate R_{21} of delay-tolerant tasks as the probability that a delay-tolerant task is processed on the UD processor locally. Here, R_{21} can be given as follows:

$$R_{21}=\sum_{i=0}^{n_{\mathrm{E}}}\sum_{j=i}^{n_{\mathrm{E}}+H_{\mathrm{E}}}\pi_{i,j,0,0}. \tag{20}$$

For a delay-tolerant task processed on the UD processor, the delay is just the service time. We call the delay of a delay-tolerant task processed on the UD processor the local delay of a delay-tolerant task. The average local delay $E[T_{21}]$ of delay-tolerant tasks can be given as follows:

$$E[T_{21}]=\frac{1}{\mu_{21}}. \tag{21}$$

Case 2: processed on the edge VMs. We define the edge offloading rate R_{22} of delay-tolerant tasks as the probability that a delay-tolerant task is processed on the edge VMs. Here, R_{22} can be given as follows:

$$R_{22}=\sum_{i=0}^{n_{\mathrm{E}}-1}\sum_{j=i}^{n_{\mathrm{E}}-1}\sum_{k=0}^{1}\sum_{r=1}^{H_{\mathrm{U}}+1}\pi_{i,j,k,r}. \tag{22}$$

For a delay-tolerant task processed on the edge VMs, the propagation delay is ignored, and the delay is just the service time. We call the delay of a delay-tolerant task processed on the edge VMs the edge delay of a delay-tolerant task. The average edge delay $E[T_{22}]$ of delay-tolerant tasks can be given as follows:

$$E[T_{22}]=\frac{1}{\mu_{22}}. \tag{23}$$

Case 3: processed on the cloud VMs. We define the cloud offloading rate R_{23} of delay-tolerant tasks as the probability that a delay-tolerant task is processed on the cloud VMs. Here, R_{23} can be given as follows:

$$R_{23}=\sum_{i=0}^{n_{\mathrm{E}}}\sum_{j=n_{\mathrm{E}}}^{n_{\mathrm{E}}+H_{\mathrm{E}}}\sum_{k=0}^{1}\sum_{r=1}^{H_{\mathrm{U}}+1}\pi_{i,j,k,r}. \tag{24}$$

For a delay-tolerant task processed on the cloud VMs, the delay includes the propagation delay from the epoch at which a delay-tolerant task departing from the EN to the epoch at which the delay-tolerant task arriving at the CDC, and the service time on the cloud VMs. We call the delay of a delay-tolerant task processed on the cloud VMs the cloud delay of a delay-tolerant task. The average cloud delay $E[T_{23}]$ of delay-tolerant tasks can be given as follows:

$$E[T_{23}]=t_{\mathrm{p}}+\frac{1}{\mu_{23}}, \tag{25}$$

where t_{p} is the average propagation delay from the EN to the CDC.

In the edge computing system with the proposed task offloading scheme, all the delay-tolerant tasks can be successfully processed. Then, R_{21}, R_{22}, and R_{23} are subject to the following condition:

$$R_{21}+R_{22}+R_{23}=1. \tag{26}$$

Combining Eqs. (21), (23), (25), and (26), we obtain the average delay $E[T_{\mathrm{dt}}]$ of delay-tolerant tasks as follows:

$$\begin{aligned}E[T_{\mathrm{dt}}]=&R_{21}\frac{1}{\mu_{21}}+R_{22}\left(t_{\mathrm{t}}+\frac{1}{\mu_{22}}\right)\\&+R_{23}\left(t_{\mathrm{p}}+\frac{1}{\mu_{23}}\right).\end{aligned} \tag{27}$$

In edge computing systems, the QoS requirements of both delay-sensitive and delay-tolerant tasks are important. Delay-sensitive tasks require high response performance. However, in practical scenarios, when the service resource is limited, higher response performance is usually achieved at the cost of lower throughput performance, and lower delay for one type of task is usually achieved at the cost of higher delay for another type of task. This brings a constrained multi-objective optimization problem. The optimization objective is to compromise different performance measures according to users' requirements.

5 Numerical studies

To numerically evaluate the long-term performance of the edge computing system with our proposed task offloading scheme, we carry out numerical experiments with analysis and simulate task offloading procedure with the generation of tasks in

the form of a Poisson process. The analysis results are presented in Matlab 2019b on Intel® Core™ i7-4790 CPU @ 3.60 GHz, 8.00 GB random access memory (RAM). The simulation results are obtained under the platform of Eclipse 4.17.0.

In state-of-the-art studies on offloading schemes, researchers focused mainly on the partial task offloading scheme (Wang ZY and Zhu, 2020), probabilistic offloading scheme (Li W and Jin, 2021), game offloading scheme (Luo and Huang, 2021), and deep reinforcement learning based offloading scheme (Song et al., 2022). In most of these strategies, tasks are usually supposed to be homogeneous and the differentiated requirements for different types of tasks are not taken into account. In our proposed offloading scheme, tasks are classified into delay-sensitive and delay-tolerant tasks according to the differentiated delay requirements. Based on the proposed three-layer edge architecture shown in Section 3, we compare the performances of the offloading scheme for heterogeneous tasks with those for homogeneous tasks.

Under the offloading scheme with homogeneous tasks, tasks are offloaded to the UD and EN by using the access threshold policy, and when both the number of tasks in the UD buffer and the number of tasks in the EN buffer reach the access threshold, the newly arriving tasks are offloaded to the CDC for processing.

In the numerical experiments and simulations, parameters are set by referring to previous works (Ranganath, 2022; Zhao et al., 2023). The parameter settings are listed in Table 3.

Table 3 Parameter settings

Parameter	Value
Total arrival rate of tasks, λ	0.1–8.0 tasks/ms
Proportion of type 1 tasks, p_1	0.5, 0.75, 1.0
Local service rate of type 1 tasks, μ_{11}	0.6 tasks/ms
Local service rate of type 2 tasks, μ_{21}	0.4 tasks/ms
Edge service rate of type 1 tasks, μ_{12}	1.0 tasks/ms
Edge service rate of type 2 tasks, μ_{22}	0.8 tasks/ms
Cloud service rate of type 2 tasks, μ_{23}	2.0 tasks/ms
Number of edge VMs in the EN, n_{E}	5
Access threshold in the UD buffer, H_{U}	5 tasks
Access threshold in the EN buffer, H_{E}	15, 20 tasks
Transmission delay from UD to EN, t_{t}	10 ms
Propagation delay from EN to CDC, t_{p}	40 ms

Type 1 and type 2 tasks denote delay-sensitive and delay-tolerant tasks, respectively. CDC: cloud data center; EN: edge network; UD: user device; VM: virtual machine

Let λ be the total arrival rate of tasks, namely, $\lambda = \lambda_1 + \lambda_2$. Let p_1 be the proportion of rate of delay-sensitive tasks to that of the total tasks generated in the UD, namely, $p_1 = \lambda_1/\lambda$. Let p_2 be the proportion of rate of delay-tolerant tasks to that of the total tasks generated in the UD, namely, $p_2 = \lambda_2/\lambda = 1 - p_1$.

Fig. 4 examines the change trend for the blocking rate R_{b} of delay-sensitive tasks in relation to the total arrival rate λ of tasks with different access thresholds H_{E} in the EN buffer and different proportions p_1 of delay-sensitive tasks.

In Fig. 4, we find that for all the access thresholds H_{E} in the EN buffer, if proportion p_1 of delay-sensitive tasks is given, the blocking rate R_{b} of delay-sensitive tasks increases as the arrival rate λ of the tasks increases. When the access thresholds H_{U} in the UD buffer and H_{E} in the EN buffer are given, the capacity of the edge computing environments is limited for delay-sensitive tasks. As the arrival rate of tasks increases, the arrival rate of delay-sensitive tasks increases accordingly. Since the service capacities of the UD and EN are limited, an increased number of delay-sensitive tasks have to wait in the EN buffer for service. The number of tasks in the EN buffer easily reaches the access threshold H_{E}; then a newly arriving delay-sensitive task is likely to be blocked. Therefore, the blocking rate of delay-sensitive tasks increases.

In Fig. 4, we also see that for the same access threshold H_{E} in the EN buffer and the same arrival rate λ of tasks, as proportion p_1 of delay-sensitive tasks increases, the blocking rate R_{b} of delay-sensitive tasks increases. When the arrival rate of tasks is given, as the proportion of delay-sensitive tasks increases, the number of tasks in the EN buffer reaches the access threshold H_E more easily. Therefore, the blocking rate of delay-sensitive tasks increases.

By comparing Fig. 4a with Fig. 4b, we find that for the same arrival rate λ of tasks and the same proportion p_1 of delay-sensitive tasks, the blocking rate R_{b} of delay-sensitive tasks with a smaller access threshold H_{E} is higher than that with a larger access threshold H_{E} in the EN buffer. The reason is that a smaller access threshold in the EN buffer makes it easier for the number of tasks in the EN buffer to reach the access threshold, and a newly arriving task is more likely to be blocked.

Fig. 5 examines how the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks changes in relation to the arrival

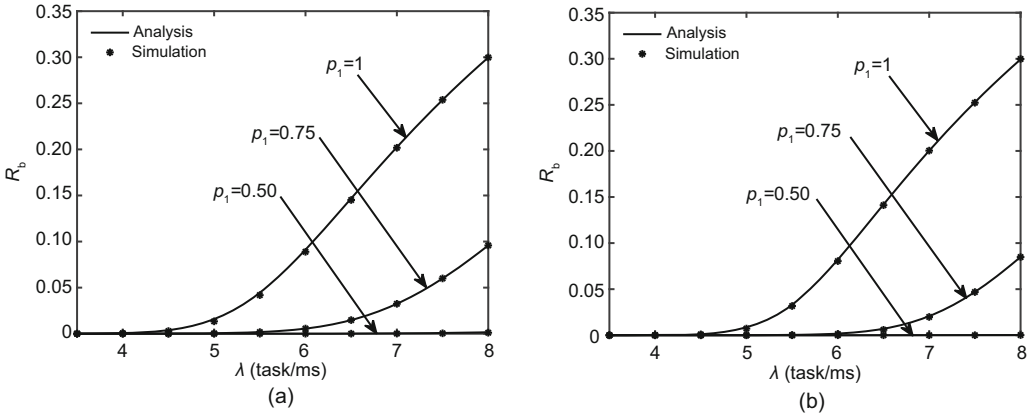

Fig. 4 Change trend for the blocking rate R_{b} of delay-sensitive tasks: (a) $H_{\mathrm{E}} = 15$; (b) $H_{\mathrm{E}} = 20$

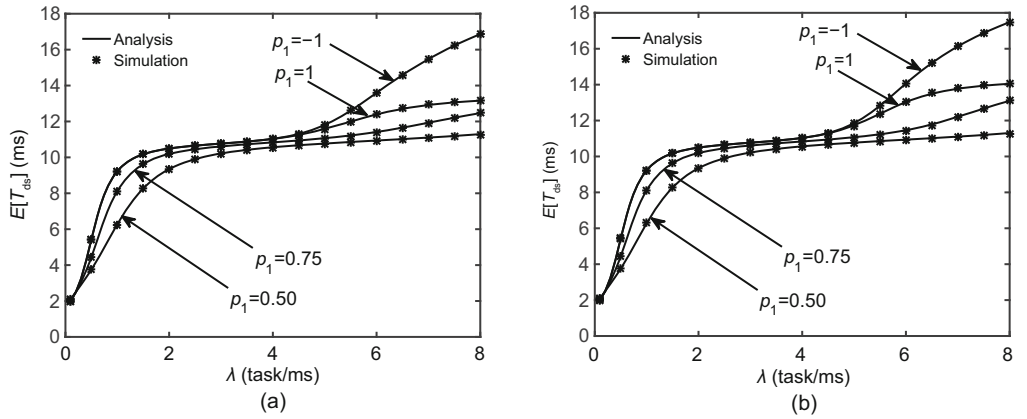

Fig. 5 Change trend for the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks: (a) $H_{\mathrm{E}} = 15$; (b) $H_{\mathrm{E}} = 20$

rate λ of tasks with different access thresholds H_{E} in the EN buffer and different proportions p_1 of delay-sensitive tasks. The lines with $p_1 = 0.50, 0.75, 1$ indicate the results from the offloading scheme with diverse tasks. The line with $p_1 = -1$ indicates the results from the offloading scheme with homogeneous tasks.

In Fig. 5, we find that for all the access thresholds H_{E} in the EN buffer, if proportion p_1 of delay-sensitive tasks is given, such as $p_1 = 0.50, 0.75, 1$, as the arrival rate λ of tasks increases, the change trend for the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks experiences three stages.

When the arrival rate λ of tasks is small, the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks shows a rising trend as the arrival rate λ of tasks increases. During this period, the number of tasks in the UD buffer is usually smaller than the access threshold H_{U}. As the arrival rate of tasks increases, a larger number of delay-sensitive tasks accumulate in the UD buffer for service. Therefore, the average delay of delay-sensitive tasks increases.

When the arrival rate λ of tasks is medium, the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks shows a smooth uptrend as the arrival rate λ of tasks increases. During this period, the number of tasks in the UD buffer reaches the access threshold H_{U} easily. As the arrival rate of tasks increases, a greater number of delay-sensitive tasks are offloaded to the EN. Considering that the service capacity of the EN is higher than that of the UD, the average delay of the delay-sensitive tasks processed on the edge VMs is lower than that processed on the UD. The relatively fixed transmission delay of delay-sensitive tasks from the UDs to the EN becomes the dominant element influencing the average delay of delay-tolerant tasks. Therefore, the average delay of delay-sensitive tasks increases smoothly.

When the arrival rate λ of tasks is high, the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks shows a rising trend again as the arrival rate λ of tasks increases. During this period, as the arrival rate

of tasks continues to increase, a greater number of delay-sensitive tasks are offloaded to the EN, and these tasks wait longer in the EN buffer for service. So, the average delay of delay-sensitive tasks processed on the edge VMs increases accordingly. For this case, the average delay of delay-sensitive tasks processed on the UD tends to be fixed. As a whole, the average delay of delay-sensitive tasks increases.

In Fig. 5, we also see that for the same access threshold H_{E} in the EN buffer and the same arrival rate λ of tasks, the change trend for the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks with a higher proportion p_1 of delay-sensitive tasks is more obvious than that with a smaller proportion p_1 of delay-sensitive tasks. When the access threshold H_{E} in the EN buffer is given, as the arrival rate of tasks increases, the higher the proportion of delay-sensitive tasks, the more intense the increase in the arrival rate of delay-sensitive tasks.

By comparing Fig. 5a with Fig. 5b, we find that for the same proportion p_1 of delay-sensitive tasks, only when the arrival rate λ of tasks is high, does the access threshold H_{E} in the EN buffer have an obvious impact on the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks. When the arrival rate of tasks is small, the delay-sensitive tasks offloaded to the EN tend to obtain service without waiting in the EN buffer, so the access threshold in the EN buffer has little impact on the average delay of delay-sensitive tasks. When the arrival rate of delay-sensitive tasks is high, most of the delay-sensitive tasks offloaded to the EN have to wait in the EN buffer before obtaining service. The higher the access threshold in the EN buffer, the longer the average waiting time of delay-sensitive tasks. This results in a higher average delay of delay-sensitive tasks.

Besides, in Fig. 5, comparing the change trends for the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks with $p_1 = 0.50, 0.75, 1$ with those with $p_1 = -1$, we find that when the arrival rate of tasks is small, the curve of $p_1 = -1$ coincides with the curve of $p_1 = 1$. However, when the arrival rate of tasks is high, the average delay of tasks with $p_1 = -1$ increases sharply. Under the offloading scheme with homogeneous tasks, when the numbers of tasks in the UD buffer and the EN buffer reach their access thresholds, the newly arriving tasks will be offloaded to the CDC, which brings a longer propagation delay. Therefore, the average delay increases. Compared to the offloading scheme with homogeneous tasks, when the traffic load of the system is higher, our proposed task offloading scheme performs better in terms of the response performance of delay-sensitive tasks.

Fig. 6 examines how the average delay $E[T_{\mathrm{dt}}]$ of delay-tolerant tasks changes in relation to the total arrival rate λ of tasks with different access thresholds H_{E} in the EN buffer and different proportions p_2 of delay-tolerant tasks. The lines with $p_2 = 0.25, 0.50, 1$ indicate the results from the offloading scheme with diverse tasks. The line with $p_2 = -1$ indicates the results from the offloading scheme with homogeneous tasks.

In Fig. 6, we find that for all the access thresholds H_{E} in the EN buffer, if proportion p_2 of delay-tolerant tasks is given, such as $p_2 = 0.25, 0.50$, and 1, as the arrival rate λ of tasks increases, the average delay $E[T_{\mathrm{dt}}]$ of delay-tolerant tasks shows three stages.

During the first stage, when the arrival rate λ of tasks is small, the average delay of the delay-tolerant tasks processed on the UD becomes the dominant element influencing the average delay of delay-tolerant tasks. Therefore, the average delay $E[T_{\mathrm{ds}}]$ of delay-sensitive tasks shows a rising trend as the arrival rate λ of tasks increases.

During the second stage, as the arrival rate of tasks increases, more delay-tolerant tasks are offloaded to the EN to obtain service. Considering that the service capacity of the EN is higher than that of the UD, the average delay of the delay-tolerant tasks processed on the edge VMs is lower than that processed on the UD. The relatively fixed and low transmission delay of delay-sensitive tasks from the UD to the EN becomes the dominant element influencing the average delay of delay-tolerant tasks. Therefore, the average delay of delay-tolerant tasks decreases.

During the third stage, as the arrival rate of tasks increases, more delay-tolerant tasks are offloaded to the CDC, and these tasks experience a longer propagation delay. For this case, the average propagation delay of delay-tolerant tasks becomes the dominant element influencing the average delay of delay-tolerant tasks. Therefore, the average delay of delay-tolerant tasks rapidly increases.

In Fig. 6, we also see that for the same access threshold H_{E} in the EN buffer and the same arrival rate λ of tasks, as proportion p_2 of delay-tolerant tasks increases, the average delay $E[T_{\mathrm{dt}}]$ of

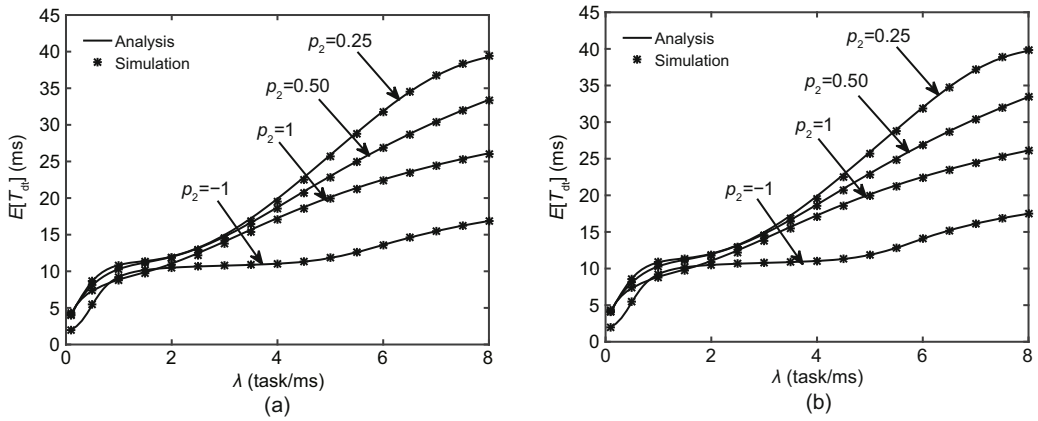

Fig. 6 Change trend for the average delay $E[T_{\mathrm{dt}}]$ of delay-tolerant tasks: (a) $H_{\mathrm{E}} = 15$; (b) $H_{\mathrm{E}} = 20$

delay-tolerant tasks decreases. As the arrival rate of tasks increases, the delay-tolerant tasks are processed mainly on the CDC. The average delay of the delay-tolerant tasks processed on the CDC is the main factor influencing the average delay of delay-tolerant tasks. As the proportion of delay-tolerant tasks increases, more delay-tolerant tasks can be processed on the UD or on the edge VMs, leading to a lower cloud offloading rate of delay-tolerant tasks. Therefore, the average delay of delay-tolerant tasks decreases.

By comparing Fig. 6a with Fig. 6b, we find that for the same proportion p_2 of delay-tolerant tasks, only when the arrival rate λ of tasks is higher, does the access threshold H_{E} in the EN buffer have some impact on the average delay $E[T_{\mathrm{dt}}]$ of delay-tolerant tasks. Besides, the higher the proportion p_2 of delay-tolerant tasks, the less the impact of the access threshold in the EN buffer on the average delay of delay-tolerant tasks. When $p_2 = 1$, the access threshold in the EN buffer has no impact on the average delay of delay-tolerant tasks. When the arrival rate of tasks is smaller, most of the tasks offloaded to the EN tend to obtain service without waiting in the EN buffer, so the access threshold in the EN buffer has little impact on the processing of tasks. When the arrival rate of tasks is higher, most of the delay-sensitive tasks offloaded to the EN have to wait in the EN buffer before obtaining service. The higher the access threshold in the EN buffer is, the longer the delay-sensitive tasks occupy the EN, and the more the delay-tolerant tasks offloaded to the CDC are. This results in a higher average delay of delay-tolerant tasks. When the proportion of delay-tolerant tasks is higher, there are fewer delay-sensitive tasks in the system, tasks in the EN buffer reach the access threshold harder, and the impact of the access threshold in the EN buffer is smaller.

Besides, in Fig. 6, comparing the change trends for the average delay $E[T_{\mathrm{dt}}]$ of delay-tolerant tasks when $p_2 = 0.25, 0.50, 1$ with that when $p_2 = -1$, we find that when the arrival rate of tasks is higher, the average delay of delay-tolerant tasks under our proposed task offloading scheme is higher than that under the offloading scheme with homogeneous tasks. When the traffic load of the system is higher, to dedicate the local and edge service resources to delay-sensitive tasks, delay-tolerant tasks are offloaded mainly to the CDC to obtain service, resulting in a higher propagation delay and a higher average delay.

Combining Figs. 5 and 6, we can see that when the traffic load of the edge computing environments is higher, our proposed task offloading scheme can improve the response performance for delay-sensitive tasks at the cost of higher delay for delay-tolerant tasks.

6 Performance optimization

From the numerical results given in Section 5, we see that the access threshold H_{E} in the EN buffer affects the average delay, the blocking rate of delay-sensitive tasks, and the average delay of delay-tolerant tasks. A lower average delay of delay-sensitive tasks can be obtained with a smaller access threshold in the EN buffer, whereas a lower blocking rate of delay-sensitive tasks can be obtained with a higher access threshold in the EN buffer. The

average delay of delay-tolerant tasks can be affected by the access threshold in the EN buffer indirectly. We conclude that there is a clear trade-off between a smaller access threshold to minimize the average delay and a higher access threshold to minimize the blocking rate.

In this study, to obtain the optimal access threshold H_{E} in the EN buffer, we construct a system cost function $F(H_{\mathrm{E}})$ based on the analytical results of the system model in Section 4.2. The system cost function $F(H_{\mathrm{E}})$ is as follows:

$$F(H_{\mathrm{E}}) = f_1 E[T_{\mathrm{r}}] + f_2 R_{\mathrm{b}} + f_3 E[T_{\mathrm{n}}], \tag{28}$$

where f_1 and f_3 are the factors for the average delays of delay-sensitive tasks and delay-tolerant tasks to the system cost respectively, and f_2 is the factor for the blocking rate of delay-sensitive tasks to the system cost. Moreover, f_1, f_2, and f_3 are all system parameters that can be set according to practical requirements.

By minimizing the system cost function $F(H_{\mathrm{E}})$, the optimal parameter H_{E}^* is given as follows:

$$H_{\mathrm{E}}^* = \underset{H_{\mathrm{E}} \geq 1}{\arg\min} \{F(H_{\mathrm{E}})\}, \tag{29}$$

where "arg min" means that the system cost will be minimized with the optimal access threshold H_{E}^* in the EN buffer.

We carry out numerical experiments to illustrate how the system cost function $F(H_{\mathrm{E}})$ changes versus the access threshold H_{E} in the EN buffer. We set $f_1 = 0.8$, $f_2 = 0.3$, $f_3 = 0.2$, $\lambda = 12$, and $p_1 = 0.45, 0.50, 0.55$; we set the other parameters to be the same as those in Table 2.

The change trend for the system cost function $F(H_{\mathrm{E}})$ versus the access threshold H_{E} in the EN buffer with different proportions p_1 of delay-sensitive tasks is shown in Fig. 7.

From Fig. 7, we find that as the access threshold H_{E} in the EN buffer increases, the system cost function experiences two stages.

During the first stage, when the access threshold H_{E} in the EN buffer is small, as the access threshold H_{E} increases, the system cost function $F(H_{\mathrm{E}})$ shows a downward trend. Then the blocking rate of delay-sensitive tasks is higher and becomes the main factor affecting the system cost function. As the access threshold in the EN buffer increases, more delay-sensitive tasks can obtain service, so the blocking rate of delay-sensitive tasks decreases. This results in a decrease in the system cost function.

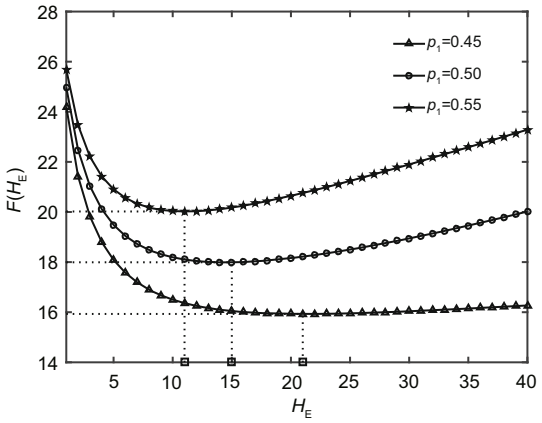

Fig. 7 System cost function $F(H_{\mathrm{E}})$ versus access threshold H_{E} in the edge network buffer

During the second stage, when the access threshold H_{E} in the EN buffer is high, as the access threshold H_{E} increases, the system cost function $F(H_{\mathrm{E}})$ shows a rising trend. Then the average delay of delay-sensitive tasks is higher and becomes the main factor affecting the system cost function. As the access threshold in the EN buffer continues to increase, the average waiting time of delay-sensitive tasks increases, so the average delay of delay-sensitive tasks increases. This results in an increase in the system cost function.

Therefore, when the access threshold is set to a proper value, there is a minimum system cost for all the proportions of delay-sensitive tasks. The minimum system cost is the value of the lowest point for each curve, and the corresponding access threshold in the EN buffer (marked with "□") is the optimal access threshold H_{E}^* in the EN buffer.

The optimal access threshold in the EN buffer and the minimum system cost with different proportions of delay-sensitive tasks are summarized in Table 4.

Table 4 Optimal access threshold in the EN buffer

Proportion p_1 of delay-sensitive tasks	Optimal access threshold H_{E}^*	Minimum system cost $F(H_{\mathrm{E}}^*)$
0.45	21	15.9309
0.50	15	17.9927
0.55	11	20.0208

From Table 4, we observe that the minimum system cost $F(H_{\mathrm{E}}^*)$ increases as proportion p_1 of delay-sensitive tasks increases. The higher the proportion

of delay-sensitive tasks, the higher the arrival rate, and the higher the blocking rate of delay-sensitive tasks. Therefore, the minimum system cost increases. The optimal access threshold H_{E}^{*} in the EN buffer decreases as proportion p_1 of delay-sensitive tasks increases. With the factor settings given in the numerical experiments of the system cost function, the access threshold in the EN buffer has less impact on the blocking rate of delay-sensitive tasks than on the average delay of delay-sensitive tasks, and a smaller access threshold in the EN buffer can lead to a lower average delay, resulting in lower system cost.

In the above experiments, we fix the parameters of the edge computing environment as a case study. In practical applications, the system parameters, such as the service rates of tasks and the factors of the performance measures, depend highly on the actual working environments. If users focus more on the average delay of delay-sensitive tasks (namely, factor f_1 increases), the impact of the average delay of delay-sensitive tasks on the system cost becomes stronger, so the access threshold H_{E} in the EN buffer should be set lower. If users focus more on the blocking rate of delay-sensitive tasks (namely, factor f_2 increases), the access threshold in the EN buffer should be set higher. For a given parameter setting, there exists an optimal access threshold in the EN buffer.

7 Conclusions

In this work, we proposed a novel task offloading scheme in the cloud–edge–device collaborative edge computing environment to meet differentiated requirements of heterogeneous tasks. With our proposed scheme, the delay-sensitive tasks can be processed on the UD or EN, and the delay-tolerant tasks can be processed on the UD, EN, or CDC. By establishing a four-dimensional CTMC as the system model and using the Gauss–Seidel method, we evaluated the long-term performance of the system. Numerical results based on the analysis and simulations show that there is a trade-off among the blocking rate of delay-sensitive tasks, the average delay of delay-sensitive tasks, and the average delay of delay-tolerant tasks when setting the access threshold in the EN buffer. For this, we constructed the system cost function and obtained the optimal access threshold in the EN buffer with different proportions of delay-sensitive tasks.

As evidenced by the results, the proposed task offloading scheme can be used for heterogeneous task scheduling in the cloud–edge–device collaboration architecture while minimizing the average delay. However, we focus only on task offloading based on stable networks from a steady-state perspective, and the task offloading scheme presented in this paper primarily emphasizes the response performance of the edge computing system. In future work, we would take energy consumption into consideration during the optimization process, thus exploring the energy–delay trade-off. Furthermore, the priority-aware dynamic offloading scheme is an intriguing yet inherently complex topic. Therefore, our research endeavors will concentrate on developing a reinforcement learning algorithm aimed at achieving the optimal solution for the energy–delay trade-off in a priority-aware dynamic offloading scheme. In addition, conducting comparison experiments with existing offloading strategies and performing comparative experiments with real cloud–edge–device collaborative systems are the focus of our next research work.

Contributors

Xiaojun BAI, Yang ZHANG, and Shunfu JIN proposed the ideas and designed the experiments. Yang ZHANG and Xiaojun BAI completed the experiments and processed the data. Xiaojun BAI, Yang ZHANG, Haixing WU, Yuting WANG, and Shunfu JIN drafted, revised, and finalized the paper.

Compliance with ethics guidelines

Xiaojun BAI, Yang ZHANG, Haixing WU, Yuting WANG, and Shunfu JIN declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

Ai LH, Tan B, Zhang JD, et al., 2023. Dynamic offloading strategy for delay-sensitive task in mobile-edge computing networks. *IEEE Int Things J*, 10(1):526-538. https://doi.org/10.1109/JIOT.2022.3202797

Akhlaqi MY, Hanapi ZM, 2023. Task offloading paradigm in mobile edge computing—current issues, adopted approaches, and future directions. *J Netw Comput Appl*,

212:103568.
https://doi.org/10.1016/j.jnca.2022.103568

Bai XJ, Jin SF, 2022. Performance analysis of an energy-saving strategy in cloud data centers based on a MMAP[K]/M[K]/N_1 + N_2 non-preemptive priority queue. *Fut Gener Comput Syst*, 136:205-220.
https://doi.org/10.1016/j.future.2022.06.004

Chahoud M, Otoum S, Mourad A, 2023. On the feasibility of federated learning towards on-demand client deployment at the edge. *Inform Process Manag*, 60(1):103150.
https://doi.org/10.1016/j.ipm.2022.103150

Djigal H, Xu J, Liu LF, et al., 2022. Machine and deep learning for resource allocation in multi-access edge computing: a survey. *IEEE Commun Surv Tutor*, 24(4):2449-2494.
https://doi.org/10.1109/COMST.2022.3199544

Feng C, Han PC, Zhang X, et al., 2022. Computation offloading in mobile edge computing networks: a survey. *J Netw Comput Appl*, 202:103366.
https://doi.org/10.1016/j.jnca.2022.103366

Gholami A, Baras JS, 2021. Collaborative cloud–edge–local computation offloading for multi-component applications. Proc IEEE/ACM Symp on Edge Computing, p.361-365.

Guo M, Wang W, Huang X, et al., 2022. Lyapunov-based partial computation offloading for multiple mobile devices enabled by harvested energy in MEC. *IEEE Int Things J*, 9(11):9025-9035.
https://doi.org/10.1109/JIOT.2021.3118016

Guo XB, Du ZL, Jin SF, 2022. Nash equilibrium and social optimization of a task offloading strategy with real-time virtual machine repair in an edge computing system. *Clust Comput*, 25(6):3785-3797.
https://doi.org/10.1007/s10586-022-03603-5

Hao YX, Jiang YY, Chen T, et al., 2019. iTaskOffloading: intelligent task offloading for a cloud–edge collaborative system. *IEEE Netw*, 33(5):82-88.
https://doi.org/10.1109/MNET.001.1800486

He JY, Zhang D, Zhou YZ, et al., 2020. A truthful online mechanism for collaborative computation offloading in mobile edge computing. *IEEE Trans Ind Inform*, 16(7):4832-4841.
https://doi.org/10.1109/TII.2019.2960127

He XQ, Shen YH, Ren J, et al., 2022. An online auction-based incentive mechanism for soft-deadline tasks in collaborative edge computing. *Fut Gener Comput Syst*, 137:1-13. https://doi.org/10.1016/j.future.2022.07.001

Hossain D, Huynh LNT, Sultana T, et al., 2020. Collaborative task offloading for overloaded mobile edge computing in small-cell networks. Proc Int Conf on Information Networking, p.717-722.
https://doi.org/10.1109/ICOIN48656.2020.9016452

Islam A, Debnath A, Ghose M, et al., 2021. A survey on task offloading in multi-access edge computing. *J Syst Archit*, 118:102225.
https://doi.org/10.1016/j.sysarc.2021.102225

Jayanetti A, Halgamuge S, Buyya R, 2022. Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge-cloud computing environments. *Fut Gener Comput Syst*, 137:14-30.
https://doi.org/10.1016/j.future.2022.06.012

Kim C, Dudin A, Dudin S, et al., 2021. Mathematical model of operation of a cell of a mobile communication network with adaptive modulation schemes and handover of mobile users. *IEEE Access*, 9:106933-106946.
https://doi.org/10.1109/ACCESS.2021.3100561

Li W, Jin SF, 2021. Performance evaluation and optimization of a task offloading strategy on the mobile edge computing with edge heterogeneity. *J Supercomput*, 77(11):12486-12507.
https://doi.org/10.1007/s11227-021-03781-w

Li YZ, Qi F, Wang ZL, et al., 2020. Distributed edge computing offloading algorithm based on deep reinforcement learning. *IEEE Access*, 8:85204-85215.
https://doi.org/10.1109/ACCESS.2020.2991773

Liao HL, Li XY, Guo DK, et al., 2022. Dependency-aware application assigning and scheduling in edge computing. *IEEE Int Things J*, 9(6):4451-4463.
https://doi.org/10.1109/JIOT.2021.3104015

Luo ZY, Huang A, 2021. Joint game theory and greedy optimization scheme of computation offloading for UAV-aided network. Proc 31st Int Telecommunication Networks and Applications Conf, p.198-203.
https://doi.org/10.1109/ITNAC53136.2021.9652130

Ma X, Wang SG, Zhang S, et al., 2021. Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing. *IEEE Trans Cloud Comput*, 9(3):968-980.
https://doi.org/10.1109/TCC.2019.2903240

Mao YY, Zhang J, Letaief KB, 2016. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE J Sel Areas Commun*, 34(12):3590-3605.
https://doi.org/10.1109/JSAC.2016.2611964

Mao YY, You CS, Zhang J, et al., 2017. A survey on mobile edge computing: the communication perspective. *IEEE Commun Surv Tutor*, 19(4):2322-2358.
https://doi.org/10.1109/COMST.2017.2745201

Muniswamaiah M, Agerwala T, Tappert CC, 2021. A survey on cloudlets, mobile edge, and fog computing. Proc 8th IEEE Int Conf on Cyber Security and Cloud Computing/7th IEEE Int Conf on Edge Computing and Scalable Cloud, p.139-142.

Ranganath S, 2022. Edge computing: types and attributes. *Adv Comput*, 127:35-62.
https://doi.org/10.1016/bs.adcom.2022.03.001

Saeik F, Avgeris M, Spatharakis D, et al., 2021. Task offloading in edge and cloud computing: a survey on mathematical, artificial intelligence and control theory solutions. *Comput Netw*, 195:108177.
https://doi.org/10.1016/j.comnet.2021.108177

Song SN, Fang ZY, Jiang JY, 2022. Fast-DRD: fast decentralized reinforcement distillation for deadline-aware edge computing. *Inform Process Manag*, 59(2):102850.
https://doi.org/10.1016/j.ipm.2021.102850

Stoyanova M, Nikoloudakis Y, Panagiotakis S, et al., 2020. A survey on the Internet of Things (IoT) forensics: challenges, approaches, and open issues. *IEEE Commun Surv Tutor*, 22(2):1191-1221.
https://doi.org/10.1109/COMST.2019.2962586

Su X, An L, Cheng Z, et al., 2023. Cloud-edge collaboration-based bi-level optimal scheduling for intelligent healthcare systems. *Fut Gener Comput Syst*, 141:28-39.
https://doi.org/10.1016/j.future.2022.11.005

Tan L, Kuang ZF, Zhao L, et al., 2022. Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing. *IEEE Trans Wirel Commun*, 21(3):1960-1972. https://doi.org/10.1109/TWC.2021.3108641

Thai MT, Lin YD, Lai YC, et al., 2020. Workload and capacity optimization for cloud-edge computing systems with vertical and horizontal offloading. *IEEE Trans Netw Serv Manag*, 17(1):227-238. https://doi.org/10.1109/TNSM.2019.2937342

Tong Z, Deng XM, Ye F, et al., 2020. Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. *Inform Sci*, 537:116-131. https://doi.org/10.1016/j.ins.2020.05.057

Vhora F, Gandhi J, 2020. A comprehensive survey on mobile edge computing: challenges, tools, applications. Proc 4th Int Conf on Computing Methodologies and Communication, p.49-55.

Wang YZ, Yu JQ, Yu ZB, 2023. Resource scheduling techniques in cloud from a view of coordination: a holistic survey. *Front Inform Technol Electron Eng*, 24(1):1-40. https://doi.org/10.1631/FITEE.2100298

Wang ZY, Zhu Q, 2020. Partial task offloading strategy based on deep reinforcement learning. Proc IEEE 6th Int Conf on Computer and Communications, p.1516-1521. https://doi.org/10.1109/ICCC51575.2020.9345003

Wu JZ, Cao ZY, Zhang YJ, et al., 2019. Edge-cloud collaborative computation offloading model based on improved partical swarm optimization in MEC. Proc IEEE 25th Int Conf on Parallel and Distributed Systems, p.959-962.

Xia SC, Wen XX, Yao ZX, et al., 2020. Dynamic task offloading and resource allocation for heterogeneous MEC-enable IoT. Proc IEEE/CIC Int Conf on Communications in China, p.847-852. https://doi.org/10.1109/ICCC49849.2020.9238863

Yang WY, Liu W, Wei XS, et al., 2021. EdgeKeeper: a trusted edge computing framework for ubiquitous power Internet of Things. *Front Inform Technol Electron Eng*, 22(3):374-399. https://doi.org/10.1631/FITEE.1900636

Zhan WH, Luo CB, Min GY, et al., 2020. Mobility-aware multi-user offloading optimization for mobile edge computing. *IEEE Trans Veh Technol*, 69(3):3341-3356. https://doi.org/10.1109/TVT.2020.2966500

Zhang JY, Yu P, Zhou FQ, et al., 2022. Resource and delay aware fine-grained service offloading in collaborative edge computing. *Comput Netw*, 218:109383. https://doi.org/10.1016/j.comnet.2022.109383

Zhang MJ, Cao JN, Yang L, et al., 2022. ENTS: an edge-native task scheduling system for collaborative edge computing. Proc IEEE/ACM 7th Symp on Edge Computing, p.149-161. https://doi.org/10.1109/SEC54971.2022.00019

Zhao H, Geng JW, Jin SF, 2023. Performance research on a task offloading strategy in a two-tier edge structure-based MEC system. *J Supercomput*, 79(9):10139-10177. https://doi.org/10.1007/s11227-023-05059-9

Zheng T, Wan J, Zhang JL, et al., 2020. A survey of computation offloading in edge computing. Proc Int Conf on Computer, Information and Telecommunication Systems, p.1-6. https://doi.org/10.1109/CITS49457.2020.9232457

Zhou WC, Fang WW, Li YY, et al., 2019. Markov approximation for task offloading and computation scaling in mobile edge computing. *Mob Inform Syst*, 2019:8172698. https://doi.org/10.1155/2019/8172698