https://doi.org/10.26599/IJCS.2022.9100031 International Journal of Crowd Science

Dynamic Task Offloading and Service Migration Optimization in Edge Networks

Yibo Han[1] ✉, Xiaocui Li[2], and Zhangbing Zhou[2,3]

ABSTRACT

In recent years, edge computing has emerged as a promising paradigm for providing flexible and reliable services for Internet of things (IoT) applications. User requests can be offloaded and processed in real time at the edge of a network. However, considering the limited storage and computing resources of IoT devices, certain services requested by users may not be configured on current edge servers. In this setting, user requests should be offloaded to adjacent edge servers or requested edge servers should be configured by migrating certain services from the former, further reducing the service access delay of user requests and the energy consumption of IoT devices in such networks. To address this issue, in this study, we model this dynamic task offloading and service migration optimization problem as the multiple dimensional Markov decision process and propose a deep q-learning network (DQN) algorithm to achieve fast decision-making, an approximate optimal task offloading, and service migration solution. Experimental results show that our algorithm performs better than existing baseline approaches in terms of reducing the service access delay of user requests and the energy consumption of IoT devices in edge networks.

KEYWORDS

task offloading; service migration; deep reinforcement learning; edge networks

Recently, the popularity of Internet of things (IoT) devices has spawned massive novel IoT applications, such as smart homes, smart cities, real-time manufacturing, and patient monitoring. These IoT applications typically show strict quality of service (QoS) requirements, such as low latency and energy consumption, which bring great challenges to the limited resources and computing power of IoT devices[1,2]. Edge computing (EC) has become the preferred platform to alleviate the conflict between low-latency requirements and the limited resources of IoT devices in emerging IoT applications[3,4]. EC supports the configuration of services on edge servers close to users and allows the offloading of complex computing tasks from user requests to nearby edge servers, thus realizing low delay and real-time processing of computing tasks[1]. In this setting, it is a key challenge to configure services to edge servers to further meet QoS constraints of user requests in edge networks. Considering the limited computation resources of IoT devices, certain edge servers may not properly configure services requested by users. In this setting, we consider the promising strategy of (1) offloading tasks to adjacent edge servers or (2) migrating some services from adjacent edge servers to the current edge server. This dynamic task offloading and service migration optimization mechanism can further reduce the service access delay of user requests and the energy consumption of IoT devices in edge networks.

The geo-distribution and limited resources of IoT devices in edge networks raise new challenges for the management of dynamic task offloading and service migration optimization. Effective algorithms have been proposed in the literature to optimize resources used by user requests to satisfy the service quality and maintain QoS constraints. Task allocation decisions play a key role in the service configuration. Here, Chen et al.[5] designed an energy-optimal dynamic computing offloading scheme through the joint optimization of various parameters. In Ref. [6], Adhikari et al. designed a priority-aware task offloading strategy based on delay dependence and assigned priority to each task according to the deadline to minimize the task offloading time. Other studies[7–10] mainly focused on partial offloading strategies or collaborative optimization of offloading decisions and resource allocation. However, most related studies only focused on task offloading to solve the delay and energy consumption limitation while ignoring the fact that users' service requests are highly dynamic in actual situations. As the edge node may not have the services required by the configuration task, this limits the task offloading policy.

Considering the conflict between the diversity of service requests and the limited number of services at edge nodes, one study proposed migrating services from the cloud to the edge of the network[11]. However, the time and energy costs of maintaining communication between a cloud server and edge nodes are very high. Therefore, frequent service migration from the cloud to the edge nodes will lead to delays and increased energy consumption. In Ref. [12], Puliafito et al. proposed that services should be migrated between edge nodes to make available services closer to user equipment and reduce the extra cost of service migration. In Ref. [13], Xu et al. solved the challenge of when and where to migrate services to achieve the best trade-off between QoS and migration cost. Meanwhile, other studies[14–16] focused on realizing low-latency service migration and seamless computing in a dynamic network environment. Task offloading and service migration problems can be modeled as a Markov decision process

1 Nanyang Institute of Big Data Research, Nanyang Institute of Technology, Nanyang 473004, China
2 School of Information Engineering, China University of Geosciences (Beijing), Beijing 100083, China
3 Computer Science Department, TELECOM SudParis, Evry 91000, France
Address correspondence to Yibo Han, hanyibo@nyist.edu.cn

(MDP), and a reinforcement learning algorithm is used to obtain the optimal strategy of MDP.

In general, the works discussed above mainly focus on task offloading or service migration. In this paper, we propose an efficient computation task offloading algorithm that dynamically combines task offloading with a service migration strategy to minimize task-processing costs. The main contributions of this paper are as follows.

(1) Based on the realistic task request mode and task cooperative processing mode, this paper analyzes the characteristics of several task request modes, sums up various request situations, and formulates corresponding migration and unloading strategies.

(2) A novel edge collaboration strategy is proposed, which considers joint task unloading and service migration, thus reducing the delay of data transmission and minimizing the overall energy consumption of IoT devices.

(3) Dynamic task offloading and service migration optimization in edge networks are modeled as a multi-dimensional MDP and then solved using the deep q-learning network (DQN) algorithm.

(4) Experimental results show that our algorithm performs better than existing baseline approaches in terms of reducing the service access delay of user requests and the energy consumption of IoT devices in edge networks.

The remainder of this paper is organized as follows. Section 1 reviews and compares the relevant techniques proposed in the literature. Section 2 analyzes and proposes the dynamic task offloading and service migration problem in edge networks and formulates it as an MDP model under given constraints. Section 3 presents a DQN-based dynamic task offloading and service migration algorithm for user requests at a one time slice to obtain an optimized service configuration scheme for maximizing the system's utility. Section 4 implements and evaluates our mechanism. Finally, Section 5 summarizes our work.

1 Related Work

1.1 Task offloading in edge computing

Task offloading plays a key role in EC. In Ref. [5], Chen et al. presented an energy-optimal dynamic computation offloading scheme by jointly optimizing various parameters. In Ref. [7], a fair task-unloading scheme is proposed while considering the energy efficiency and priority of IoT devices. In Ref. [8], Wang et al. introduced a new dynamic EC model and designed an online primal-dual algorithm to offload the arrival tasks. To improve the generalization ability of the task-offloading algorithm, Wang et al.[9] proposed a task-unloading method based on meta-reinforcement learning. Considering the limited resources of IoT devices, one study proposed a resource-aware adaptive task offloading framework and flexibly selected the optimal unloading strategy[10]. Generally, these algorithms greatly inspire us to develop the dynamic task offloading and service migration mechanism. However, these works did not consider the possibility that services demanded by user requests may not be configured on the requested node. In this paper, a dynamic task offloading and service migration is introduced, which takes the intermediate nodes with abundant computation resources as the execution nodes of tasks. If the requested node is not configured with the corresponding service, the service migration is carried out.

1.2 Service migration in edge computing

Limited resources of IoT devices and delay-sensitive constraints of user requests bring more challenges to task offloading and service migration in EC. Introducing service migration between IoT devices to achieve low delay and seamless computing has attracted much attention. In Ref. [14], services were deployed on edge nodes in advance according to the user activity hotspot map, enabling Zhang et al.[14] to introduce a service migration strategy based on user activity hotspots. In Ref. [15], Liang et al. developed an optimal iterative solution method based on relaxation and rounding to minimize the migration cost. In Ref. [16], Kim et al. modeled the cost minimization of service migration as an integer linear programming problem, which is difficult to solve due to resource constraints at the servers and unknown user mobility patterns, and proposed alternative heuristic solution algorithms that performed well in both theory and practice. At the same time, reinforcement learning is becoming a promising method to achieve rapid convergence. In Ref. [17], Miao et al. proposed that the decision-making process of service migration should be modeled as a one-dimensional Markov decision and that reinforcement learning can be used to optimize the service migration decision-making. To sum up, these techniques have inspired us to develop the method used in the current paper. However, they can hardly be used directly in our context. Therefore, we propose an efficient computation task offloading and service migration algorithm, wherein deep reinforcement learning is used to achieve rapid decision-making.

2 System Model

This section introduces the system model, including the network model, energy consumption model, and delay model.

2.1 Network model

Edge networks can be modeled as a set of N edge nodes expressed as $F = \{F_1, F_2, F_3, \ldots, F_i, \ldots, F_N\}$, where F_i is described as $F_i = (\mathrm{id}, \mathrm{Loc}, \mathrm{HostLst}, \mathrm{PT}, \mathrm{Fre})$. Here, id_{F_i} is the unique identifier of the i-th edge node F_i, $\mathrm{Loc}_{F_i} = (L_x, L_y)$ is the geographical location of F_i, and L_x and L_y are the longitude and latitude of the geographical position of F_i, respectively. In addition, $\mathrm{HostLst}_{F_i}$ is the list of services currently hosted by F_i, PT_{F_i} is the list of tasks currently being processed by the edge node F_i, and Fre_{F_i} is the processor frequency of F_i. Meanwhile, M services are deployed in edge networks, which can be denoted as $S = \{S_1, S_2, S_3, \ldots, S_i, \ldots, S_M\}$, where $S_i = (\mathrm{id}, \mathrm{Byt}, \mathrm{RCT})$, id_{S_i} is the unique identifier of the i-th service S_i, Byt_{S_i} is the workload size of S_i, and RCT_{S_i} is the service configuration time of S_i. In this paper, at one time slice t, K user requests are issued and denoted as $C = \{C_1, C_2, C_3, \ldots, C_i, \ldots, C_K\}$, where $C_i = (\mathrm{id}, \mathrm{Loc}, S_j, \mathrm{Byt}, \mathrm{Cr})$. In addition, id_{C_i} is the unique identifier of the i-th user request C_i; $\mathrm{Loc}_{C_i} = (L_x, L_y)$ is the geographical location of C_i, and L_x and L_y are the longitude and latitude of the geographical position of C_i, respectively; $S_{j_{C_i}}$ denotes the service used to execute C_i; Byt_{C_i} is the data size of C_i; and Cr_{C_i} is the number of required central processing unit (CPU) cycles of C_i.

2.2 Energy model

For each task request C_i, the total energy consumption is denoted as follows:

$$E_{\mathrm{total}} = E_{\mathrm{comp}} + E_{ti} + E_{tij} \tag{1}$$

where E_{comp} is the computing and processing energy consumption of C_i, E_{ti} is the energy consumption of offloading C_i to edge node F_i, and E_{tij} is the energy consumption of migrating one service S_j from F_j to F_i. Actually, E_{ti} and E_{tij} are the communication energy

consumption generated by sending data packets from one edge node to another. According to the first-order radio model, the communication energy consumption of sending k-bit data from F_i to F_j with distance d can be expressed as

$$E_{\text{cmm},ij}(k,d) = E_{\text{Tx}}(k,d) + E_{\text{Rx}}(k) \tag{2}$$

where

$$E_{\text{Tx}}(k,d) = E_{\text{elec}} \times k + \varepsilon_{\text{amp}} \times k \times d^n \tag{3}$$

$$E_{\text{Rx}}(k) = E_{\text{elec}} \times k \tag{4}$$

in which $E_{\text{Tx}}(k,d)$ is the transmission energy by forwarding k-bit data from F_i to F_j with distance d, and $E_{\text{Rx}}(k)$ represents the energy of receiving k-bit data by F_j. The value of the attenuation index n is greatly influenced by the environment. If edge nodes in the network are barrier-free when forwarding data packets, the value of n can be set to 2; otherwise, it is set to 3, 4, or 5. In addition, E_{elec} and ε_{amp} represent the unit energy consumption constant and the unit energy consumption constant of the transmitting amplifier, respectively. The related parameters of the communication energy consumption model are listed in Table 1.

2.3 Delay model

For each task request C_i, the total delay is denoted as follows:

$$D_{\text{total}} = D_{\text{comp}} + \text{Max}\left(D_{ti}, \text{ArgMin}\left\{D_{fji}\right\} + D_r\right) + D_w \tag{5}$$

where D_{ti} is of offloading C_i to edge node F_i, D_{fji} is the time to migrate one service S_j from F_j to F_i, D_r is the configuration time of S_j, and D_w is the waiting time of C_i. The above D_{ti} and D_{fji} are the communication time caused by sending data packets from one edge node to another. D_{comp} is the computing and processing time of C_i, which is defined as

$$D_{\text{comp}} = \frac{\text{Cr}_C}{\text{Fre}_F} \tag{6}$$

In this paper, the communication delay is defined as follows:

$$\text{delay} = k_{\text{net}} \log_{10} d_{ij} + b_{\text{net}} \tag{7}$$

where d_{ij} is the distance between F_i and F_j. In addition, k_{net} and b_{net} are represented, respectively, as follows:

$$k_{\text{net}} = 44.9 - 6.55 \log_{10} h_b \tag{8}$$

$$b_{\text{net}} = 46.3 + 33.9 \log_{10} f - 13.82 \log_{10} h_b - \text{ah}_m + c_m \tag{9}$$

where f is the signal frequency, h_b is the antenna height of F_i, and ah_m is defined as follows:

$$\text{ah}_m = 3.20\left(\log_{10}(11.75 h_r)\right)^2 - 4.97 \tag{10}$$

2.4 Problem formulation

In this paper, we intend to shorten the execution time of user requests, as shown in Eq. (5) and save the energy consumption of each edge node, as presented in Eq. (1). The formalized problem is defined as follows:

$$\min C \tag{11}$$

where

$$C = \left(\omega_1 \sum_{i=1}^{k} D_i^{\text{total}} + \omega_2 \sum_{i=1}^{k} E_i^{\text{total}}\right) \tag{12}$$

where D_i^{total} is the total time for executing all user requests; E_i^{total} is the total energy consumption for completing all user requests; ω_1 and ω_2 are the weight parameters of execution time and energy consumption, respectively; and $\omega_1 + \omega_2 = 1$.

Table 1 Parameters used in the first-order radio model.

Parameter	Description
E_{elec}	Unit energy consumption constant
ε_{amp}	Energy consumption constant of unit transmitting amplifier
n	Propagation attenuation index
k	Data size (bit)
d	Data transmission distance

3 Dynamic Task Offloading and Service Migration Mechanism

3.1 Task offloading status space

The system status space is based on delay, energy consumption, service type (ST), running status of edge node, and all user requests at one time slice t. The total delay and energy consumption can be expressed as follows:

$$X(t) = \begin{bmatrix} x_{1,1}(t) & x_{1,2}(t) & \cdots & x_{1,N}(t) \\ x_{2,1}(t) & x_{2,2}(t) & \cdots & x_{2,N}(t) \\ \vdots & \vdots & \ddots & \vdots \\ x_{K,1}(t) & x_{K,2}(t) & \cdots & x_{K,N}(t) \end{bmatrix} \tag{13}$$

where $x_{k,n}(t)$ is the total delay of the k-th user request executed upon the n-th edge node at one time slice t.

$$Y(t) = \begin{bmatrix} y_{1,1}(t) & y_{1,2}(t) & \cdots & y_{1,N}(t) \\ y_{2,1}(t) & y_{2,2}(t) & \cdots & y_{2,N}(t) \\ \vdots & \vdots & \ddots & \vdots \\ y_{K,1}(t) & y_{K,2}(t) & \cdots & y_{K,N}(t) \end{bmatrix} \tag{14}$$

where $y_{k,n}(t)$ is the total energy consumption of the k-th user request executed upon the n-th edge node at t.

Generally, the system state at one time slice t can be expressed as

$$S_t = \left\{X(t), Y(t), \text{HostLst}_F, \text{PT}_F, \text{Loc}_C, S_C\right\} \in S \tag{15}$$

3.2 Service migration action space

The system action space is defined as the candidate set of edge nodes to execute user requests at one time slice t. The system action at T_i can be expressed as a one-dimensional vector as follows:

$$A_t = \left\{a^{F_n}, a^{\text{CL}}\right\} \in A \tag{16}$$

where $a^{F_n} = \left\{a^{F_1}, a^{F_2}, \ldots, a^{F_N}\right\}$ represents whether the k-th user request C_k is executed or not. When a^{F_n} is 1, C_k is executed on edge node F_n. Here, F_n is the edge node, and CL is the cloud node.

3.3 Reward function

The reward function at T_i of the system is formulated by

$$R_t = M_1 - C \tag{17}$$

where M_1 is a constant.

3.4 Task offloading and service migration algorithm

Algorithm 1 describes the DQN algorithm. For each episode, the

Algorithm 1 DQN-based task offloading and service migration

Require:

θ: Weight of Q network

RP: Experience replay pool of Q network

Ensure:

A_t: A strategy set of the dynamic task offloading and service migration for user requests at one time slice t

1: for episode=1, 2, …, P do
2: initialize S_1 and RP
3: for κ=1, 2, …, t do
4: $A_\kappa \leftarrow$ actionSelection (S_κ, Q_m)
5: Observe $S_{\kappa+1}$
6: $R_\kappa \leftarrow$ calculated by Eq. (17)
7: $\theta \leftarrow$ Q-NetworkUpdate $(S_\kappa, A_\kappa, S_{\kappa+1}, R_\kappa, \text{RP})$
8: end for
9: end for
10: return A_t

initial state S_1 is obtained. Algorithm 1 performs iteratively to select A_κ using a predefined strategy, obtain the next state $S_{\kappa+1}$ and R_κ by Eq. (17), as presented in Algorithm 2, and update Q network (Lines 1–9), as presented in Algorithm 3. Finally, A_κ is returned (Line 10).

Algorithm 2 describes the specific action selection procedure and mainly includes two parts: explorer and developer. A random value Φ is generated during each selection. When Φ is greater than ε, the action is selected by the developer; otherwise, the action is selected by the explorer (Lines 1–7).

Algorithm 3 describes the Q network update process. For each time, $(S_\kappa, A_\kappa, R_\kappa, S_{\kappa+1})$ are stored in the experience replay pool RP, and a sample RP_κ is randomly extracted from RP to update the Q network (Lines 1–13).

4 Performance Evaluation

In this paper, our experimental environment is constructed using a 64-bit Windows10 operating system, 16 GB memory, Intel (R) Core (TM) i7-3770 CPU @ 3.40 GHz, and NVIDIA GeForce GTX 1070 Ti, which is realized by Python program. To ensure the scientific nature of the experiment, all the experiments were conducted in the same operating environment.

Algorithm 2 Action selection

Require:

S_κ: Current state

Q_m: Best state-action table

Ensure:

A_κ: A selecting action set

1: generate Φ randomly
2: if $\Phi > \varepsilon$ and $S_\kappa \neq$ null, $S_\kappa \in Q_{m_i}$ then
3: $A_\kappa \leftarrow \text{argmax}(S_\kappa, A_\kappa; \theta)$
4: else
5: $A_\kappa \leftarrow$ Randomly select F_i from the list of available computing resources L
6: end if
7: return A_κ

Algorithm 3 Q network update

Require:

S_κ: Current state

A_κ: Current action

R_κ: Current reward

$S_{\kappa+1}$: Next time slice state

Replay pool (RP): Experience replay pool of Q network

Ensure:

θ: Weight of Q network

1: Store $(S_\kappa, A_\kappa, R_\kappa, S_{\kappa+1})$ in RP
2: Sample RP_κ from RP
3: for $(S^j_{\kappa-1}, A^j_{\kappa-1}, R^j_{\kappa-1}, S^j_\kappa)$ do
4: Calculate $Q(S^j_\kappa, A^j_\kappa; \theta')$
5: Calculate $y_d(\kappa)$ by Eq. (14)
6: if v = null, $v \in Q_{m_i}$ or $y(\kappa) > v \in Q_{m_i}$ then
7: $v \in Q_{m_i}, S_\kappa \leftarrow y_d(\kappa)$
8: $a \in Q_{m_i} \leftarrow A^j_\kappa$
9: end if
10: end for
11: $\theta \leftarrow \text{argmin}_\theta L(\theta)$
12: from time to time reset $\theta' \leftarrow \theta$
13: Return θ

4.1 Experimental setup

In this paper, the experimental data are generated by simulating a 500 m × 500 m network, where 50 edge servers are randomly deployed, and their communication radius is set by 150 m. Two kinds of services are configured upon the edge servers. The other parameters are described in Table 2.

4.2 Experimental results and evaluation analysis

By observing the long-term average delay and energy consumption of the system, this experiment explores the influence of different system parameters, such as the sparsity of user requests and the number of STs.

4.2.1 Sparsity of user requests

When the sparsity of user requests is higher, more user requests will be generated at each time on average, and the average response delay and energy consumption of the system will be higher. To study the influence of user request sparsity on average delay and energy consumption, the user request sparsity (prob) is set as 0.2, 0.4, and 0.8 in the experiments. Figure 1 shows the trend of system average response delay with different prob. The average response delay of the system is directly proportional to the number of user requests at each moment. When prob = 0.4 and prob = 0.2, the average response delay of the system is not doubled. The reason is that when prob = 0.2, the computation load of the network is not saturated, and thus user requests can be

Table 2 Experimental parameters.

Experimental parameter	Value
Network area size	500 × 500 m²
Number of edge nodes N	50
Maximum number of services configured on one edge node	2
Communication radius of one edge node	150 m
Edge CPU frequency Fre_F	2 GHz
Maximum number of user requests K at one time slice	30
Requested sparsity prob	0.4
Number of network service types (Snum)	50
Service data size Byt_S	[0.2 MB, 5 MB]
Number of CPU cycles required for the task Cr_C	[50, 200]
Amount of data for the task Byt_C	[0.5 MB, 5 MB]
Transmission bandwidth w_i of wireless signals	500 MHz
Signal frequency f	2.5 MHz
Antenna height h_b	35 m
User height h_r	1 m
Energy consumption constant E_{elec}	50 nJ/bit
Energy consumption constant $\varepsilon_{\mathrm{amp}}$ of the transmission amplifier	0.1 nJ/(bit·m²)
Calculation of energy consumption of CPU in a single cycle	6×10^{-9} W·h
Learning rate α	0.1
Discount rate γ	0.9

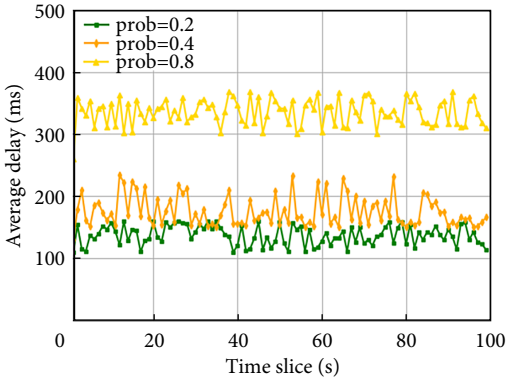

Fig. 1 Average delay with different rates of user request sparsity.

executed. However, when prob = 0.4, the computation load of the network is close to saturation. User requests cannot be processed immediately at certain suitable edge nodes. When prob = 0.8, the computation load of the network is supersaturated, and the average delay is doubled compared with other prob settings.

Figure 2 shows the trend of system average energy consumption with different prob values. Intuitively, the average energy consumption of the system is directly proportional to the number of user requests. When the computation load of the system is close to saturation, the task offloading and service migration are more frequent, and thus the cooperative processing of these nodes will inevitably lead to an increase in transmission energy consumption. Therefore, the average energy consumption of the system increases linearly with the sparsity of user requests.

4.2.2 Number of service types

Figure 3 shows the average delay of the system with the number of

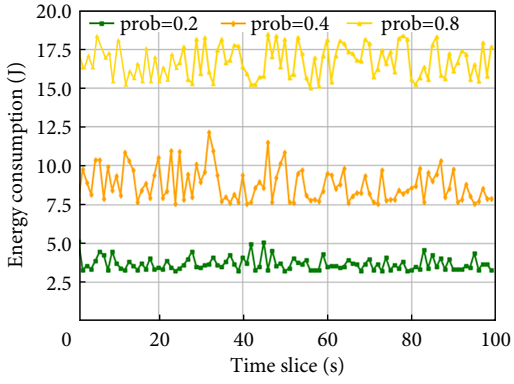

Fig. 2 Average energy consumption with different rates of user request sparsity.

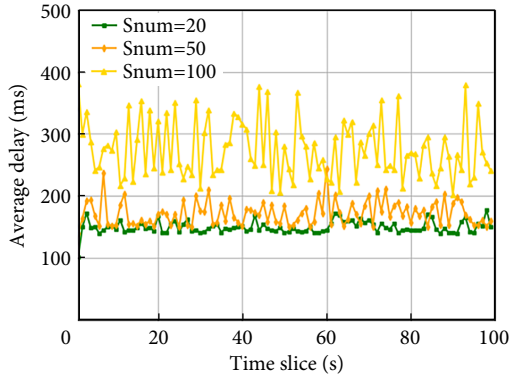

Fig. 3 Average delay with different numbers of service types.

STs, where the prob is set as 0.4. As can be seen, when ST is set as 20, all STs can find multiple edge nodes configured with this ST in the edge network. Thus, there are more candidate edge nodes for task offloading and service migration, and it is easier to find task-processing locations adjacent to the data generation locations. Therefore, when the computation load of the network is not completely saturated, the average response delay of the system is smaller as ST decreases. Notably, when ST is set as 100, the volatility of the average delay of the system increases significantly. This means that the type of user request may not find edge nodes configured with corresponding services, the edge nodes configured with corresponding services may have insufficient computing resources, or their physical distance is not close enough. In this case, edge nodes must offload tasks to the cloud or migrate ST in the cloud to the edge layer to ensure the effective processing of tasks. Such long-distance migration or offloading will lead to a sharp delay increase. Thus, when the number of STs is far more than the number of services that can be configured at edge nodes, our DQN-based dynamic task offloading and service migration algorithm can properly address the pre-configure services and ensure the richness of the services at the edge layer.

Figure 4 shows the influence of the number of different STs on the average energy consumption of the system. As can be seen, compared with the average delay of the system, when the number of ST is set as 100, the system average energy consumption volatility of the edge network is larger than when the number of ST is set as 20 and 50 due to the reason presented in Fig. 3.

4.3 Comparable evaluation with TO-DQN, TO-GD, CS-NO, and CR-GD

To demonstrate the superiority of the proposed DQN-based dynamic task offloading and service migration algorithm in terms

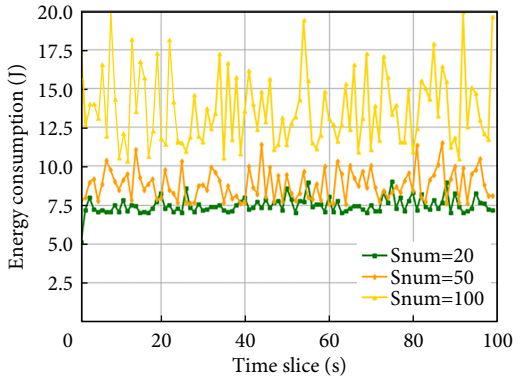

Fig. 4 Average energy consumption with different numbers of ST.

of the service access delay of user requests and energy consumption of IoT devices, we compare it with four baseline algorithms as follows.

- **TO-DQN:** This algorithm adopts DQN to optimize configuration strategies and aims to offload tasks to edge servers without considering the migration of some services to the requested edge server.
- **TO-GD:** This algorithm adopts the Greedy to optimize configuration strategies and aims to offload tasks to edge servers without considering the migration of some services to the requested edge server.
- **CS-NO:** This algorithm merely offloads tasks to the cloud.
- **CR-GD:** This algorithm randomly offloads tasks to both edge servers and the cloud.

This comparative experiment observed the total response delay and total energy loss of the system under different energy weights of the above algorithms with fewer than 100 time slices. The comparison results are respectively shown in Figs. 5 and 6.

4.3.1 Average delay evaluation

(1) By comparing CR-DQN and TO-DQN with the CR-GD and TO-GD groups, respectively, we can see that the total time delay of the system can be effectively reduced by using the DQN algorithm when making location selection decisions. The result is achieved regardless of the collaborative processing mode centered on computing resources proposed in this paper or the traditional task-unloading mode. As analyzed in Section 2, greedy decision-making for current task requests can obtain the optimal solution. However, in the long-term continuous decision-making process, this optimal liberation is likely to become a local optimal or even a poor solution because this algorithm does not fully consider the

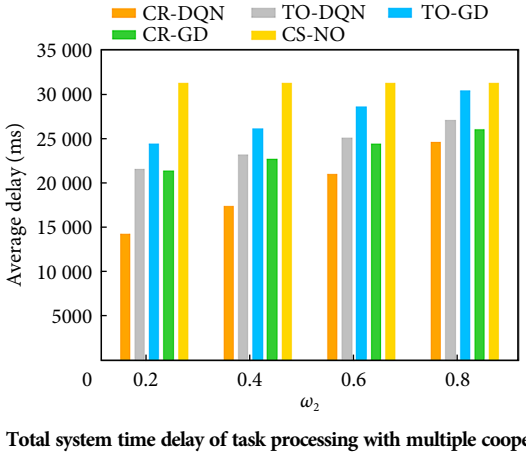

Fig. 5 Total system time delay of task processing with multiple cooperative processing strategies under 100 time slices with different energy weights.

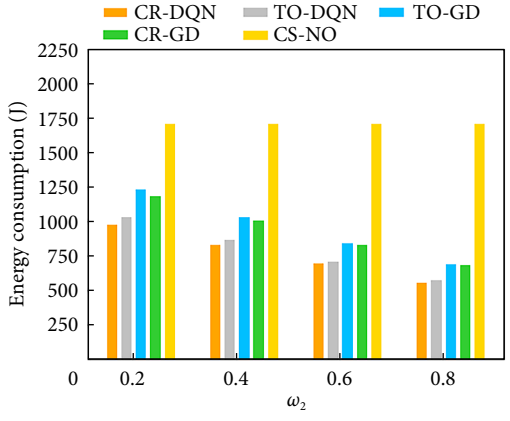

Fig. 6 Total energy consumption of the system with various cooperative processing strategies under 100 time slices with different energy weights.

changes in the network environment and the randomness of user requests in the future time slice. The DQN algorithm can learn from the past unloading experience to a certain extent to adapt to the complex and changeable network environment. Therefore, in the long-term decision-making process, the DQN algorithm can achieve a better solution compared with the other alternatives.

Notably, we can see in Fig. 5 that, after adopting the DQN algorithm, the total system delay of CR-GD is reduced more significantly than that of TO-GD. The reason for this phenomenon is that the CR strategy has more selectable fog nodes than the TO strategy, and in extreme cases, the selectable fog nodes list of the latter degenerates into local fog nodes or neighboring fog nodes (in this case, the DQN algorithm). However, the selectable fog nodes of the CR strategy are always globally available. When the computational load is not supersaturated, the selected space scale is guaranteed, so the DQN algorithm can more fully mine the globally optimal solution.

(2) By comparing CR-DQN and CR-GD with the TO-DQN and TO-GD groups, respectively, we can see that when the same selection decision algorithm is adopted, the collaborative processing strategy centered on computing resources can effectively reduce the total response delay of the system compared with the traditional task-unloading method. Similar to analysis (1) in Section 4.3.1, it can also be observed from Fig. 5 that the CR strategy can significantly reduce the total response delay of the system when the DQN algorithm is adopted. Given that the CR strategy does not directly pay attention to the configuration of services and task requests, it only pays attention to the currently available computing resources. Thus, it provides a richer list of selectable fog nodes, which also means that the scale of its problems is larger. To ensure a fast search for feasible solutions, the GD algorithm may sacrifice the global optimality of decision-making, while the DQN algorithm memorizes the past efficient solutions with the help of a deep neural network, thus allowing the rapid searching of high-quality solutions in a large-scale solution space. It can be seen that both the CR strategy and the DQN algorithm can achieve good results in the task cooperation processing problem in fog computing. Furthermore, the large-scale solution space problem caused by the CR strategy can also be solved by DQN. Thus, the combination of these two algorithms can achieve more remarkable results in solving this problem.

(3) Observing the experimental group CS-NO, we find that the total delay of the system under this cooperative processing mode is significantly higher than that of the other groups. The reason for this phenomenon is that when this cooperative processing mode is used alone, the cloud vertically migrates the service to the local

fog node for service reconfiguration once there is no ST corresponding to the configuration request locally. Obviously, the cost of communication with the cloud is high, which also means that the response delay of task processing will inevitably increase.

4.3.2 Energy consumption evaluation

(1) By comparing CR-DQN and TO-DQN with the CR-GD and TO-GD groups, respectively, we can see that using the DQN algorithm can effectively reduce the total energy consumption of the system for the same reason as analysis (1) in Section 4.3.1.

(2) By comparing CR-DQN and CR-GD with the TO-DQN and TO-GD groups, respectively, it can be found that the CR strategy cannot significantly reduce the total energy consumption of the system. As can be seen in Eq. (1), when this strategy is implemented, the main energy loss depends on the distance of data packet transmission. In this task-processing strategy involving computing resources, the available computing resources are selected as relay points for task processing, and service migration and task unloading are carried out simultaneously. In the experimental setup of this paper, the data size range of services is close to that of tasks. To simplify the analysis, this paper assumes that the packet size of the service and task is the same, so the main energy loss only depends on the transmission distance, and the sum of the distances between the moving service to the relay point and the unloading task to relay point is similar. Thus, this strategy is obviously unable to reduce the total energy consumption of task processing in most cases. However, in the actual process of task request processing, the size of the service packet is different from that of the task packet. Therefore, when making a decision, we can weigh the size of the service packet and task packet to determine whether the relay point of task execution is biased toward the service-providing side or the task request side to reduce energy consumption.

(3) By observing the experimental group CS-NO in Figs. 5 and 6, it can be found that the cost increase caused by the CS-NO cooperation mode is more significant in the total energy consumption of the system. The reason for this phenomenon is that there are large STs in the experimental setup of this paper. Thus, for this type, if services are always migrated, it means a great deal of communication energy loss, and CS strategy is a potential cooperative processing mode. It has, however, shown good performance in lightweight service migration. Therefore, when using this strategy, it is necessary to fully consider the data size and reconfiguration time of the service itself to carry out reasonable service migration.

5 Conclusion

In this paper, we propose a DQN-based dynamic task offloading and service migration strategy that leverages the collaboration of edge nodes with limited resources, such as computation, storage, and bandwidth, to reduce the delay and energy consumption in the process of user requests. According to the proposed strategy, the state space, action space, and reward function of the deep reinforcement learning algorithm are set, and the decision of task-processing place is made by using deep reinforcement learning. In doing so, we can solve the problem of not being able to use traditional heuristic algorithms in time-delay-sensitive scenes due to their high complexity.

To verify the feasibility and efficiency of the strategy proposed in this paper, the influences of two different network parameters on the strategy are explored, and two traditional cooperative task-processing methods are compared. The experimental results reveal that the strategy proposed in this paper can effectively reduce the system task-processing delay and reduce the system energy consumption to a certain extent. Experimental results further show that our algorithm performs better than baseline approaches in terms of reducing the service access delay of user requests and the energy consumption of IoT devices in edge networks.

Dates

Received: 11 January 2022; Revised: 8 September 2022; Accepted: 11 September 2022

References

[1] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, An application placement technique for concurrent IoT applications in edge and fog computing environments, *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1298–1311, 2021.

[2] E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig, Foggy: A framework for continuous automated IoT application deployment in fog computing, in *Proc. 2017 IEEE International Conference on AI Mobile Services*, Honolulu, HI, USA, 2017, pp. 38–45.

[3] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, Towards edge intelligence: Multi-access edge computing for 5G and internet of things, *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6722–6747, 2020.

[4] C. S. Yang, R. Pedarsani, and A. S. Avestimehr, Edge computing in the dark: Leveraging contextual-combinatorial bandit and coded computing, *IEEE/ACM Transactions on Networking*, vol. 29, no. 3, pp. 1022–1031, 2021.

[5] S. Chen, Y. Zheng, W. Lu, V. Varadarajan, and K. Wang, Energy-optimal dynamic computation offloading for industrial IoT in fog computing, *IEEE Transactions on Green Communications and Networking*, vol. 4, no. 2, pp. 566–576, 2020.

[6] M. Adhikari, M. Mukherjee, and S. N. Srirama, DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing, *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5773–5782, 2020.

[7] G. Zhang, F. Shen, Y. Yang, H. Qian, and W. Yao, Fair task offloading among fog nodes in fog computing networks, in *Proc. 2018 IEEE International Conference on Communications*, Kansas City, MO, USA, 2018, pp. 1–6.

[8] H. Wang, H. Xu, H. Huang, M. Chen, and S. Chen, Robust task offloading in dynamic edge computing, *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 500–514, 2021.

[9] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, Fast adaptive task offloading in edge computing based on meta reinforcement learning, *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2021.

[10] H. Tran-Dang and D. -S. Kim, FRATO: Fog resource based adaptive task offloading for delay-minimizing IoT service provisioning, *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 10, pp. 2491–2508, 2021.

[11] C. Lin and H. Khazaei, Modeling and optimization of performance and cost of serverless applications, *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 615–632, 2020.

[12] C. Puliafito, E. Mingozzi, and G. Anastasi, Fog computing for the internet of mobile things: Issues and challenges, in *Proc. 2017 IEEE International Conference on Smart Computing*, Hong Kong, China, 2017, pp. 1–6.

[13] J. Xu, X. Ma, A. Zhou, Q. Duan, and S. Wang, Path selection for seamless service migration in vehicular edge computing, *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 9040–9049, 2020.

[14] R. Zhang, L. Wen, S. Naboulsi, T. Eason, V. K. Vasudevan, and D. Qian, Accelerated multiscale space—Time finite element simulation and application to high cycle fatigue life prediction, *Computational*

Mechanics, vol. 58, no. 2, pp. 329–349, 2016.

[15] Z. Liang, Y. Liu, T. -M. Lok, and K. Huang, Multi-cell mobile edge computing: Joint service migration and resource allocation, *IEEE Transactions on Wireless Communications*, doi: 10.1109/TWC.2021.3070974.

[16] T. Kim, S. D. Sathyanarayana, S. Chen, Y. Im, X. Zhang, S. Ha, and C. Joe-Wong, MoDEMS: Optimizing edge computing migrations for user mobility, in *Proc. 2022 IEEE International Conference on Computer Communications*, London, UK, 2022, pp. 1159–1168.

[17] Y. Miao, F. Lyu, F. Wu, H. Wu, J. Ren, Y. Zhang, and X. S. Shen, Mobility-aware service migration for seamless provision: A reinforcement learning approach, in *Proc. 2022 IEEE International Conference on Communications*, Seoul, Republic of Korea, 2022, pp. 5064–5069.