Zheng *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2022) 11:3
https://doi.org/10.1186/s13677-021-00276-0

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Deep Reinforcement Learning-Based Workload Scheduling for Edge Computing

Tao Zheng[1,2*], Jian Wan[1], Jilin Zhang[1] and Congfeng Jiang[1]

Abstract

Edge computing is a new paradigm for providing cloud computing capacities at the edge of network near mobile users. It offers an effective solution to help mobile devices with computation-intensive and delay-sensitive tasks. However, the edge of network presents a dynamic environment with large number of devices, high mobility of users, heterogeneous applications and intermittent traffic. In such environment, edge computing often suffers from unbalance resource allocation, which leads to task failure and affects system performance. To tackle this problem, we proposed a deep reinforcement learning(DRL)-based workload scheduling approach with the goal of balancing the workload, reducing the service time and the failed task rate. Meanwhile, We adopt Deep-Q-Network(DQN) algorithms to solve the complexity and high dimension of workload scheduling problem. Simulation results show that our proposed approach achieves the best performance in aspects of service time, virtual machine(VM) utilization, and failed tasks rate compared with other approaches. Our DRL-based approach can provide an efficient solution to the workload scheduling problem in edge computing.

Keywords: Edge computing, Computation offloading, Workload scheduling, Deep reinforcement learning

Introduction

Nowadays, with the increasing popularity of mobile devices, more novel sophisticated applications are emerging, such as face recognition, inter-active gaming and augmented reality [1]. However, duo to resource constraints (processing power, battery lifetime, storage capacity), mobile devices cannot meet the needs of running these novel sophisticated applications on local [2]. Considering the powerful computing and storage capabilities of the cloud server, one suitable solution is to offload these complicated mobile applications to cloud for processing, so called mobile cloud computing (MCC) [3]. MCC can efficiently address the problems of limited processing capabilities and limited battery of the mobile devices [4]. However, the cloud server is generally far away from mobile devices, MCC inevitably suffers from high latency and bandwidth limitation [5]. Moreover, according to the prediction of Cisco, the growth rate of mobile data required to be processed will far exceed the capacity of central clouds in 2021 [6]. To resolve these issues, Edge computing has emerged as a promising technology that provides cloud computing capabilities at the edge of the network in close proximity to the mobile subscribers [7]. Compared with MCC, edge computing has the advantage of lower latency, lower core network load and more security.

Although Edge computing is a new technology with many advantages, it still has many problems to be solved. The edge of network presents a very dynamic environment with large number of devices, high mobility of users, heterogeneous applications and intermittent traffic [8]. In such environment, edge computing always encounters the problem of how to efficiently schedule incoming tasks from mobile devices to edge servers and cloud servers. To elaborate, the mobile devices consist of terminal devices and Internet of Things (IoT), which are widely distributed, numerous heterogeneous and high mobility. When these

*Correspondence: zhengtao@zjsru.edu.cn
[1]School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China
[2]College of Information Science and Technology, Zhejiang Shuren University, Hangzhou 310015, China

mobile devices are running, they can generate a variety of different tasks. Duo to the resource constraint of mobile devices, most of tasks need to be offloaded to the outside servers for processing. However, these offloaded tasks are unevenly distributed and random, which lead to imbalanced workloads among edge servers and impair the performance of system. For example, when massive amount of tasks are offloaded to the same edge server simultaneously, it is easy to cause single edge server paralysis and network congestion, while other edge servers may be in idle state. Therefore, how to schedule the incoming stream of offloaded tasks determines the overall efficiency and scalability of the edge computing system. Moreover, both the communications and computation resources also need to be allocated and scheduled efficiently for better system performance.

Our work focuses on the workload scheduling problem which can be defined as deciding on the destination computational unit for each offloaded task within an edge computing system. As we know, workload scheduling is an intrinsically hard, online problem [8]. Where to offload decision should take many parameters into consideration such as task property, network situation and computational resources [9]. These parameters are also highly dynamic especially under unexpected variation of the load. To solve problem, we propose DRL-based workload scheduling approach, which can learn from the previous actions and achieve best scheduling in the absence of a mathematical model of the environment. Meanwhile, we adopt DQN-based algorithms to solve the complexity and high dimension of workload scheduling problem. Finally, we carry out an experimental evaluation based on EdgeCloudSim [10]. To demonstrate the performance of our approach, we evaluated with two opponent algorithms using crucial performance metrics such as service time, failed tasks rate, and VM utilization. According to the results, our proposed method has competitive performance with respect to its opponents for the cases studied. The contributions of this paper are summarized as follows:

We investigate workload scheduling in edge computing, aiming at balancing the workload, reducing the service time and failed task rate.

We introduce multi-tier edge computing architecture and analyze its system model, then forumlate workload scheduling problem as an NP-hard problem.

We proposed a DRL-based workload scheduling approach with the goal of balancing the workload, reducing the service time and the failed task rate.

We adopt DQN-based algorithms to solve the problem of complexity and high dimension in workload scheduling.

The remainder of this paper is organized as follows. Related work section briefly reviews the related work. System model and problem formulation section presents the system model and problem formulation. The DQN-based workload scheduling approach section describes the proposed DRL-based workload scheduling approach. Evaluation results section elaborates on the simulation experiment design and analyzes the results. Conclusion section concludes our study and provides possible directions for future research.

Related work

In edge computing, Mobile devices can offload most tasks to the edge server for execution, which efficiently address the problems of their limited resources and reduce the core network traffic load. However, improper task offloading not only brings imbalance workload among edge servers, but also increases task latency and energy consumption. Therefore, properly scheduling computation tasks among edge servers are crucial to optimize the quality of services with high resource efficiency. The scheduling research is to choose the proper decision on the time and place where the task should be offloaded. There has been extensive work devoted to workload scheduling problem. Santoro et al. [11] propose a software platform called Foggy for workload orchestration and resource negotiation in fog computing environment. It schedules the execution location of tasks based on computational, storage or network resources. Anas et al. [12] take computational utilization and access probability into consideration, and develop a performance model based on queuing theory to address the workload balancing between service providers within a federated cloud environment. Ma et al. [13] consider cooperation among edge nodes and investigate the workload scheduling with the objective of minimizing the service response time as well as the outsourcing traffic in mobile edge computing. They propose a heuristic workload scheduling algorithm based on water-filling to reduce computation complexity. Fuzzy logic is an effective method to solve workload scheduling problem in edge computing, which has been discussed in recent years. Sonmez et al. [8] employ a fuzzy logic-based approach to solve the workload orchestration problem in the edge computing systems. Their approach takes into consideration the properties of the offloaded task as well as the current state of the computational and networking resources, and uses fuzzy rules to define the workload orchestration actions in terms of networking, computation and task requirements to decide the task execution location within the overall edge computing system.

However, Fuzzy logic-based approach needs to define various fuzzy rules in advance, which will take a lot of time and effort to measure. Therefore, in order to reduce manual intervention, some studies attempt to adopt machine learning method for workload scheduling. Nascimento, A., et al [14] propose Reinforcement Leaning(RL)-based scheduling approach for cloud-based scientific workflow

execution. RL is a branch of machine learning, which focuses on how to achieve the optimal goals through learning in a dynamic environment. In RL, the agent, as a learner, perceives the current state of environment and select actions to take. When the action is executed, the agent will receive a reward or punishment from environment according to the effect of action. If receiving a reward from the environment, the agent will increase the tendency to take this action in the future to obtain more rewards. On the contrary, if receiving a punishment, the agent will reduce the tendency to take this action. To maximize the cumulative reward, agent needs to balance the exploration and exploitation steps. In exploration step, the agent tries the actions that have not been selected before and explore new state to obtain higher reward. In exploitation step, the agent takes the best action that has already observed so far [15].

Although the RL-based workload scheduling approach can reduce manual intervention and solve the problem effectively in the case of small state and action spaces, It is nearly impossible to obtain the accurate state or action values via normal reinforcement learning when the state or action spaces are very large [16]. To solve this problem, Combining deep learning and reinforcement learning, DRL algorithms, such as DQN [17], DDPG [18] and PPO [19], becomes useful for handling the problems of complexity and high dimension. In this work, We proposed a DRL-based approach for workload scheduling in edge computing, which can learn from the previous actions and achieve best scheduling in the absence of a mathematical model of the environment, Meanwhile, we adopt DQN algorithms to solve the workload scheduling problem of complexity and high dimension, aiming at balancing the workload among edge servers, reducing the service time and failed task rate.

System model and problem formulation

In this section, we first introduce the Multi-tier edge computing architecture and analyze system model, which includes task model, network model, and computing model. Then, we formulate the workload scheduling problem based on system model. The edge computing architecture as depicted in Fig. 1

Multi-tier edge computing architecture

As shown in Fig. 1, we construct a multi-tier edge computing system architecture which incorporates computational resources at various levels, as well as different ranges of networks, such as local area network (LAN), metropolitan area network (MAN) and wide area network

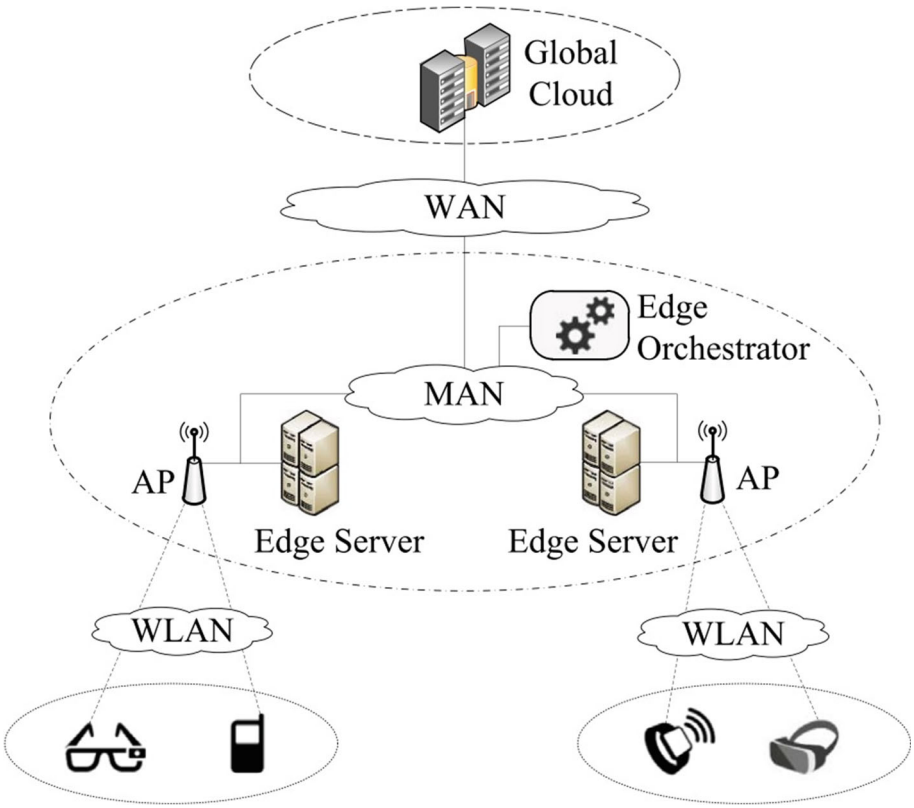

Fig. 1 Multi-tier edge computing architecture with edge orchestrator

(WAN). The first tier is edge device layer, which is composed of a large number of mobile devices, IoTs and other edge devices. They communicate with local edge server via wireless local area network (WLAN). The second tier is edge server layer, which consist of edge servers and the edge orchestrator. Edge servers are interconnected through MAN. The third tier is cloud server layer, which is the global cloud server distributed around world and provide cloud service through WAN.

In this architecture, each edge server can provide computing services for users within its WLAN coverage, while cloud server can provide remote computing services for all users. Moreover, nearby edge servers can also provide computing services for neighbor LAN users in domain. When an edge server cannot provide sufficient resources or computation capacity, the computation tasks can be scheduled to the nearby edge servers that are under-utilized or the cloud for processing. In this process, the edge orchestrator acts as the workload coordinator among servers to monitor the environment information and make scheduling decisions for each task to coordinate workload of servers.

To elaborate, we take the university campus application scenario as an example. According to this architecture, the students and other participants are assumed to carry and/or wear the mobile devices, such as smart phone, Google glasses, Smart Bracelet, etc., on which applications run and continuously generate tasks to be processed. Duo to the limited processing capacity of a single device, these devices have to offload some tasks to external servers. To meet this need, a number of edge servers are deployed at different places (such as classroom buildings, dormitories, and library) on campus to provide coverage for request services. In the real scene, the places on campus may have different user densities according to the time of day. For example, students concentrated their classes in the classroom buildings in the morning, and they may gather together in the library for reading in the afternoon, and most of them are likely stay in dormitories at night. The user density can directly affect the amount of requested and workload. Typically, the user's mobile devices select the nearest local edge server to offload tasks via the WLAN, such as Wi-Fi hotspots or cellular base stations. In our system, we define the places with three type attractiveness levels, which are based on user's device density. If the attractiveness level of the place is high, such as an auditorium during the conference, lots of students may wear the mobile device and gather there. Duo to the high device density and excessive task offloading, both the network and the computational resources are likely to be congested in LAN. To deal with this problem, the local edge server also needs to offload these tasks which beyond its capacity to the nearby edge servers or cloud server. However, how to schedule these tasks to achieve the desired optimization objectives is not an easy problem.

System model

To investigate workload scheduling problem, we should first model the edge computing system according to the above architecture, including task model, network model and computing model. We set that the system consists of k mobile devices, m edge servers and one remote cloud server. k and m denote the number of mobile devices and edge servers respectively. For each device, there is only one local edge server which can be accessible via WLAN and $(m-1)$ nearby edge severs which can be accessible via MAN. Each task offloaded from mobile device can be scheduled to execute on local edge server, nearby edge server, or cloud server. In general, the local server is the first choice due to the advantages of short distance and low latency. However, when local edge server cannot provide sufficient resources or computing power, the corresponding tasks should be scheduled to the nearby edge servers or cloud for processing. In this process, the edge orchestrator acts as the workload coordinator among servers to monitor the environment information and make scheduling decisions for each task to coordinate workload of servers.

1) Task Model

Each mobile device can produce different tasks, which may be compute-intensive, I/O intensive or both. To better represent the tasks, we describe a $task_i$ in a four-field notation $(\partial_i, \beta_i, c_i, \tau_i)$, which ∂_i denotes the input data size (in bits), β_i denotes the output data size (in bits), and c_i denotes the total number of CPU cycles that is required to complete the $task_i$, and τ_i denotes the delay constraint of $task_i$. We assume that all the tasks are indivisible and offloaded as a whole to edge server or cloud for execution, and the tasks are independent of each other.

2) Network Model

As shown in Fig. 1, In our edge computing system, tasks can be scheduled to execute on any server, such as local edge server, nearby edge servers or cloud server. Therefore, tasks may pass through multiple networks during scheduling process, such as WLAN, MAN or WAN. Considering the difference of bandwidth, transmission rate and interference among different networks, these will have a certain impact on the transmission delay and performance of task scheduling. We calculate the communication delay generated by scheduling tasks to local edge server, nearby edge servers and cloud server, respectively: If the local edge server is selected as offloading destination, the mobile device first uploads the computing to the local edge server through a wireless channel, and then edge server returns the result to the mobile device after the task is completed. In this process, the communication

delay on WLAN is mainly caused by transmission delay of $task_i$, which can be expressed as:

$$\begin{aligned} t^i_{com} &\approx t^i_{td} = t^i_{up} + t^i_{down} \\ t^i_{up} &= \frac{\partial_i}{R_{wlan}}; \quad t^i_{down} = \frac{\beta_i}{R_{wlan}}; \end{aligned} \tag{1}$$

where, t^i_{com} represent the communication delay of $task_i$, t^i_{td} represent the transmission delay of $task_i$, t^i_{up} represent the upload time of $task_i$, and t^i_{down} represent the result download time of $task_i$,R_{wlan} represent WLAN transmission rate. In our system, we assume that the upload and download transmission rate between mobile devices and local edge server are the same. The WLAN transmission rate is quantified as:

$$R_{wlan} = \omega_{wlan} \log_2 \left(1 + \frac{p_n h_{n,s}}{N_0 + \sum_{n=1}^{M} p_n h_{n,s}} \right) \tag{2}$$

Where, ω_{wlan} is the bandwidth of WLAN, p_n is the transmission powers of edge device, $h_{n,s}$ is the channel gains between the n_{th} mobile device and the s_{th} edge server, N_0 is the noise power.

According to the above formulas, we can see that the WLAN transmission rate is closely related to the bandwidth. For example, if too many mobile devices choose the same channel to communicate with the base station and the interference between them will increase, which will affect the transmission rate.

If the nearby edge server is selected as offloading destination, the $task_i$ will be scheduled to the nearby edge server via the MAN, the transmission delay is mainly affected by the MAN bandwidth. The transmission delay of the $task_i$ on MAN can be expressed as:

$$t^i_{td} = t^i_{up} + t^i_{down} = \frac{\partial_i + \beta_i}{R_{man}} \tag{3}$$

Where, R_{man} represent the MAN transmission rate.

If the remote cloud server is selected as offloading destination, the $task_i$ will be uploaded to the cloud server via the WAN. In general, there is a long distance between the cloud server and the user, the propagation delay of signal back and forth cannot be ignored. Therefore, the communication delay includes transmission delay t_{td} and propagation delay t_{pd}. The communication delay of the on WAN can be expressed as:

$$\begin{aligned} t^i_{com} &= t^i_{td} + t^i_{pd} \\ &= t^i_{up} + t^i_{down} + t^i_{pd} \\ &= \frac{\partial_i + \beta_i}{R_{wan}} + t^i_{pd} \end{aligned} \tag{4}$$

The propagation delay is much longer than the transmission delay ($t_{pd} \gg t_{td}$). Thus, the communication delay is mainly determined by the propagation delay.

3) Computing Model

For workload scheduling, an important indicator is service time, which represents the time it takes for the task to upload to the server until it is completed. Service time consists of task waiting time and execution time. In this section, we will calculate the service time required on the local edge server, nearby edge server and the cloud, respectively. When a task is uploaded to the local edge server, it first will be generally arranged in the task queue of server to wait for processing. There is a waiting time between the uploading to server and starting execution. The waiting time is related to the VM utilization of the local edge server. If VM utilization is low, which means that the current local edge server is relatively idle, the task can get VM resources quickly and its waiting time will be short. On the contrary, if VM utilization is high, which means that the current local edge server is relatively busy, and the waiting time will be long. Thus, the service time of $task_i$ on local edge server can be expressed as:

$$t^i_s = t^i_{wait} + t^i_{exe}, \quad t^i_{exe} = \frac{c_i}{f_l} \tag{5}$$

Where,c_i is the total number of CPU cycles that is required to complete the $task_i$. f_l represents local edge server computing power. t^i_{wait} is waiting time between the uploading to local edge server and starting execution

Similarly, when the task is migrated to the nearby edge server, the service time of $task_i$ on nearby edge server can be expressed as

$$t^i_s = t^i_{wait} + t^i_{exe}, \quad t^i_{exe} = \frac{c_i}{f_{nb}} \tag{6}$$

Where, f_{nb} represents nearby edge server computing power. t^i_{wait} represents waiting time between the uploading to nearby edge server and starting execution. When the $task_i$ is offloaded to the cloud, we consider that cloud resources are always sufficient, the waiting time of tasks can be ignored. Thus, the service time on cloud sever can be expressed as

$$t^i_s \approx t^i_{exe} = \frac{c_i}{f_c} \tag{7}$$

Where, f_c represents cloud computing power. According to the above analysis, as we know that the main factors that affect the service time are the amount of computation required for the task, VM utilization and computing power of the server.

4) Definition of task Failure Each task has its own delay constraint τ, if the running time exceeds this constraint value, the task is considered to be failure. We define the condition for task failure as follows:

$$F = \left\{ task_i \,\middle|\, t^i_{com} + t^i_{exe} > \tau_i, i \in N \right\} \tag{8}$$

Problem formulation and analysis

Based on the analysis of the task, network and computing model, we calculator the total delay of task scheduling to local edge server, nearby edge server and cloud server respectively. The total delay on local edge server:

$$\begin{aligned} t^i_{local} &= t^i_{com} + t^i_s \\ &= t^i_{up} + t^i_{wait} + t^i_{exe} + t^i_{down} \\ &= \frac{\partial_i + \beta_i}{R_{wlan}} + \frac{c_i}{f_l} + t^i_{wait} \end{aligned} \tag{9}$$

The total delay on nearby edge server:

$$\begin{aligned} t^i_{nb} &= t^i_{com} + t^t_s \\ &= t^i_{up} + t^i_{wait} + t^i_{exe} + t^i_{down} \\ &= \frac{\partial_i + \beta_i}{R_{man}} + \frac{c_i}{f_{nb}} + t^i_{wait} \end{aligned} \tag{10}$$

The total delay on cloud server:

$$\begin{aligned} t^i_{cloud} &= t^i_{com} + t^t_s \\ &= t^i_{up} + t^i_{exe} + t^i_{down} + t^i_{pd} \\ &= \frac{\partial_i + \beta_i}{R_{wan}} + \frac{c_i}{f_c} + t^i_{pd} \end{aligned} \tag{11}$$

According to total delay of local edge server, nearby edge server and cloud server. Our optimization objective of the workload scheduling problem is to reduce the total delay and failure rate of the task. The optimization problem is formulated as follows:

$$\begin{aligned} \min\ t &= \sum_{i=1}^{n} \lambda_1 t^i_{local} + \lambda_2 t^i_{nb1} ... \lambda_m t^i_{nb(m)} + \lambda_{m+1} t^i_{cloud} \\ s.t\ \ & t^i_{com} + t^i_{exe} < \tau_i \\ & \lambda_l, \ldots \lambda_{m+1} \in \{0, 1\} \end{aligned} \tag{12}$$

Where, $\lambda_1, \ldots \lambda_{m+1}$ represent the scheduling decision variables. Since tasks are indivisible, these scheduling decision variable are integer variables of 0 and 1, but only one decision variable can be 1 in each decision, which represents the selected offloading destination. For example, If $\lambda_1 = 1$ and the others are 0, which means the local edge server is selected as the offloading destination. If $\lambda_i = 1,\ (i \in [2, m])$, which means the i_{th} nearby edge server is selected. If $\lambda_{m+1} = 1$ and the others are 0, which means the cloud is selected as offloading destination. Since the decision variables are integer variables, the optimal solution can be found by traversing when the number of tasks is small. However, with the increasing of tasks, the scale of solution space will increase rapidly and become too large. In this case, Equation (12) is no longer a convex optimization problem, but a NP-hard. Moreover, the effect of task scheduling is affected by many parameters, such as network bandwidth, VM utilization, and task attributes. These parameters can be highly dynamic especially under unexpected variation of the load. In the next subsection, we propose a workload scheduling approach based on deep reinforcement learning to solve the problem.

The DQN-based workload scheduling approach

According to the above analysis, we conclude that workload scheduling in edge computing is an online and NP-hard problem. In this section, we will introduce our proposed DRL-based approach for workload scheduling problem.

Specifically, the DQN is one of DRL algorithms in order to tackle the complex and high dimension problem. The DQN modeling is based on the Markov decision processes (MDP), which includes State space S, action space A, and reward function R. In order to apply DQN in the edge computing scenario, we need to specifically design different components of MDP as described below:

State space

In DRL approach, the first step is to define the state of the system environment. As far as we know, our multi-tier edge computing system has thousands of time-varying parameters at runtime. Therefore, how to select key parameters to accurately describe the current system environment is crucial to task scheduling. According to the previous analysis, the VM utilization status of the servers and the network conditions from users to servers should be considered. These parameters can be highly dynamic especially under unexpected variation of the workload. Moreover, the characteristics of task also play an important role on the system performance [7]. In our work, the state space S contains all of environment states, and each state includes three parameters: server states, network states and tasks characteristic.

$$\begin{aligned} S &= \{s_1, s_2 \cdots s_n\} \\ s_t &= \{server_t, network_t, task_t\}, s_t \in S \end{aligned} \tag{13}$$

Where, $server_t$ represents the states of all servers at step t: $server_t = \{u_1, u_2 \cdots u_n\}$, the u_n is the VM utilization of n_{th} server. The $network_t$ is the states of networks at step t: $network_t = \{\omega^t_{WAN}, \omega^t_{MAN}, \omega^t_{WLAN}\}$, the ω^t_{WAN}, ω^t_{MAN} and ω^t_{WLAN} represent the bandwidth of WAN,MAN and WLAN at step t, respectively. $task_t$ is the task that need to be scheduled at step t : $task_t = \{\partial_t, \beta_t, c_t, \tau_t\}$, which ∂_t denotes the input data size (in bits), β_t denotes the output data size (in bits), and c_t denotes the total number of CPU cycles that is required to complete the task, and τ_t denotes the corresponding delay constraint of $task_t$.

Action space

The next step is to define the action space A. Actions determine the transfer rules between states. When a task

arrives, it needs to decide which the suitable server should be choice to execute the task. Therefore, the action space includes all servers. In our system, the server consists of local edge server, multiple nearby edge servers and a cloud server. In nearby edge servers, we select the nearby edge server with the least loaded. At the decision step t, the action space can be simplified as follows:

$$A_t = \left\{a_l, a_n^1 \ldots a_n^{m-1}, a_c\right\} \tag{14}$$

Where, a_l represent the action that allocate task to the local edge server, $a_n^1 \ldots a_n^{m-1}$ represent the action that allocate task to one of the nearby edge severs, (m-1) represent the number of nearby edge servers, a_c represent the action that allocate task to the cloud server.

Reward function

The reward function is used to describe the immediate reward from one state to another state after an action is taken, based on which to evaluate the action. For a reward function, the first thing to do is to determine what the optimization goal is, and judge the positive reward or negative reward according to its goal.

In this paper, our optimization goal is to reduce the total delay and failure rate of the task. As far as we can see, the main reason of task failure is the unreasonable task allocation, which leads to the task waiting for execution or the transmission time is too long and exceed its deadline. Therefore, we can evaluate the reward according to following formula:

$$R_t = \frac{(\tau_t - T_t)}{\frac{1}{n}\sum_{1}^{n}|(\tau_i - T_i)|} \tag{15}$$

where, T_t represents the total delay of current task, τ_t represents the corresponding delay constraint of $task_t$. First of all, we judge whether to reward or punish by whether the task can be completed within the deadline. If $\tau_t > T_t$,which means the current task is completed within its deadline and receive a positive reward. If $\tau_t < T_t$, which means the current task exceed its deadline and receive a negative reward or punishment. The size of the reward is measured by reference to the average.

DQN-based workload scheduling algorithm

Based on the components defined above, our goal is to balance the workload and reduce the service time. Our algorithm scheme is shown in Fig. 2. We adopt two neural networks in DQN, one is called the online network and the other is called target network, they have the same structure but different parameters. Online network uses the latest parameters θ_i, which are updated according to the loss Function. The target network uses parameters θ_i^-, which are copied from the online network at every N step.

DQN uses a deep convolution network to calculate the action-value function of online network, which can be expressed as $Q(s_t, a_t, \theta_i)$. The action-value function of target network can be expressed as:

$$y_i = r_t + \gamma \max_a Q\left(s_{t+1}, a, \theta_i^-\right) \tag{16}$$

where,R_t represents the reward received after taking action a_t, $\gamma \in \{0, 1\}$ is the discount factor, which is used for function converge. the max Q-value action a is selected to execute in the current state s_t and transfer to the next state s_{t+1}.

In the training of DQN, the loss function of each experience is defined as the mean square error between the value function of online network and target network:

$$L_i(\theta_i) = \mathrm{E}\left[\left(y_i - Q(s_t, a_t, \theta_i)^2\right)\right] \tag{17}$$

Take the partial derivative of parameter and the gradient is as follows:

$$\nabla_{\theta_i} L(\theta_i) = E\left[\left(y_i - Q(s, a, \theta_i)\right) \nabla_{\theta_i} Q(s, a, \theta_i)\right] \tag{18}$$

DQN reduces the loss to a limited range through gradient descent, and both the value function and the gradient value are in the normal range, which ensures the stability of the algorithm.

To perform experience replay we store the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time-step t in replay memory $D_t = (e_1, \ldots, e_t)$. During training, some empirical data are randomly selected from the replay memory D to update the parameters of the network by stochastic gradient descent. The advantage of this experience playback mechanism can break data correlations and make neural network update more efficient.

Our DQN-based Workload Scheduling algorithm is presented in Algorithm 1, which is divided into six steps and we describe each step as follow:

Step 1: Input variables and initialize parameters. The variables include $N_{\min}$, $N_{\max}$, Δ , α , γ , ε . $N_{\min}$ is the min number of mobile devices, $N_{\max}$ is the max number of mobile devices, Δ represents the number of devices increased each time. Thus, the number of iterations can be calculated as $M = \frac{N_{\max} - N_{\min}}{\Delta}$; we also need to set the learning rate α, the discount factor γ and the greedy coefficient ε.

Step 2: Initialize replay memory D to capacity N, which is used to store experience sample(transition) for experience replay, then initialize action-value function Q with random weights θ and target action-value function $\hat{Q}$ with weights $\theta = \theta^-$.

Step 3: Adopt the ε-greedy strategy to select action a_t according to the state s,then execute action a_t and observe reward r_t and next state s_{t+1} to store transition (s_t, a_t, r_t, s_{t+1}) into replay memory D

Step 4: After each network output, start training by randomly sampling a number of Batch data from replay

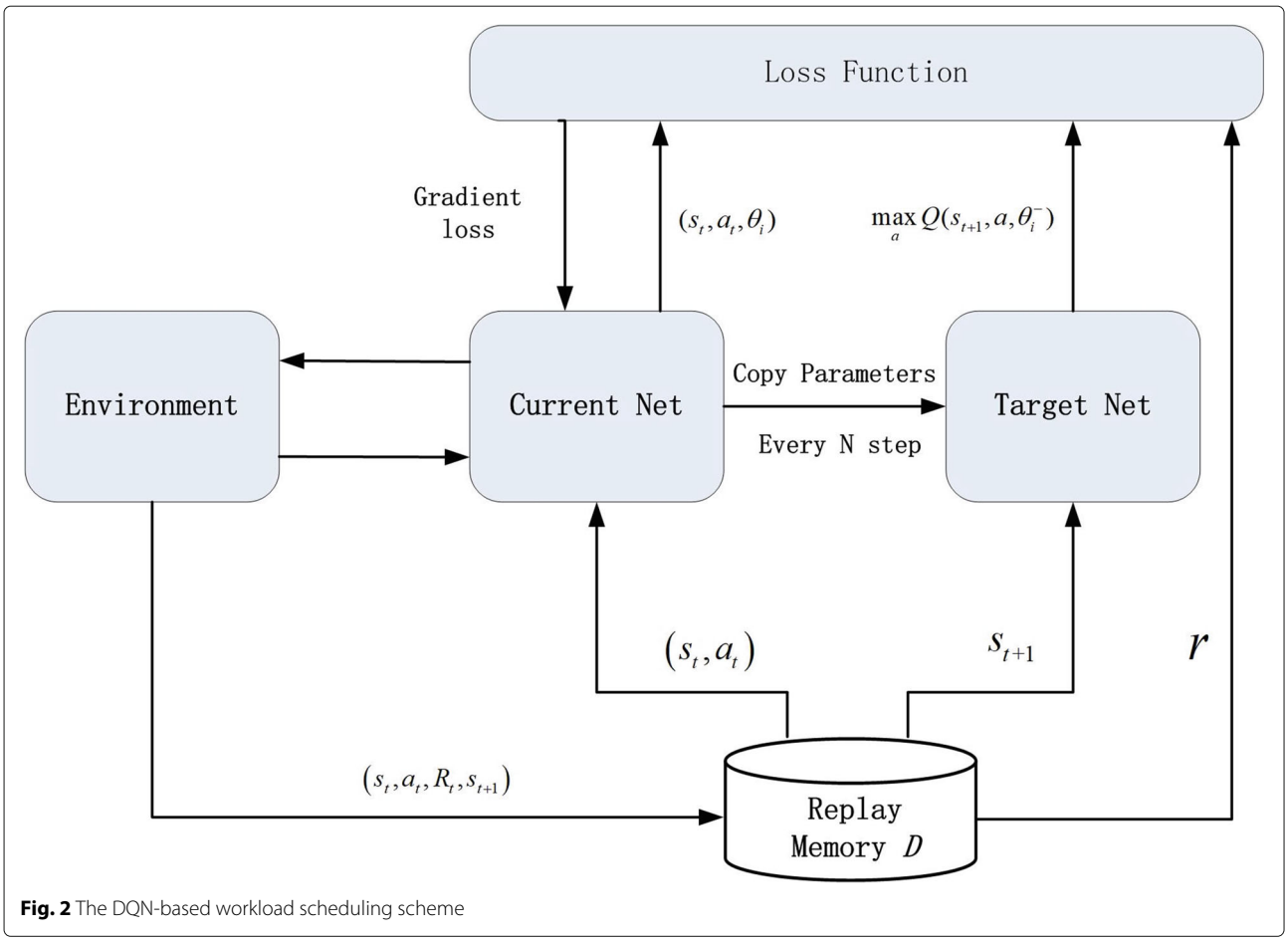

Fig. 2 The DQN-based workload scheduling scheme

memory D, update the action value of online network. Set target network output according to equation(16).

Step 5: Perform a gradient descent step on equation (17) with respect to the network parameters.

Step 6: After several steps of each iteration, copy parameter θ of the online network to the target network as the new parameter θ^-.

Evaluation results

In this section, we conducted a large number of experiments to evaluate the performance of proposed DQN-based workload scheduling algorithm. To illustrate the effectiveness of our algorithm, we compare our algorithm with other algorithms in term of the average number of failed tasks, average service time, average VM utilization and average network delay. Before presenting the evaluation results, we first introduce the competitor algorithms and evaluation setup.

Competitor algorithms

In order to benchmark the proposed DQN-based solution, we compare it with another two DRL algorithms, which are the DDPG-based and PPO-based workload scheduling algorithms.

1) The DDPG-based algorithm combines the actor-critic method, deterministic strategy gradient method, and experience playback mechanism to learn the optimal strategy in model-free continuous control. Action network and criticism network are used in DDPG online decision stage, and experience replay and target network are used in offline training stage. The DDPG-based workload scheduling algorithm is presented in Algorithm 2.

2) The PPO-based algorithm is under Actor-Critic structure and alternate between sampling data through interaction with the environment, and optimizing a surrogate objective function using stochastic gradient ascent [19]. The core idea of PPO is to restrict the update range of the old and new policies to improve stability [20]. The PPO-based Workload Scheduling Algorithm is presented in Algorithm 3.

Evaluation setup

In this work, all experiments are performed on the EdgeCloudSim [10], which provide a simulation environment

Algorithm 1 DQN-based Workload Scheduling Algorithm

Input: $N_{\min}, N_{\max}, \Delta, \alpha, \gamma, \varepsilon$

Output: workload scheduling decision

Initialize replay memory D to capcaity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta = \theta^{-}$

For episode = 1,M do

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi_1(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    Otherwise select $a_t = \max_a Q(s_t, a, \theta)$

    Execute action a_t and observe reward r_t and next state x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in repaly memory D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from repaly memory D

    Set $y_j = \begin{cases} r_j \\ r_j + \gamma \max_a \hat{Q}(s_{j+1}, a', \theta^{-}) \end{cases}$

    Perform a gradient descent step on $(y_j - Q(s_j, a_j, \theta))^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End for

End for

Online making workload scheduling decision:

Load the parameters θ;

Calculate action-value $Q(s_t, a; \theta)$

Output a_t = argmax$Q(s_t, a; \theta)$

Algorithm 2 DDPG-based Workload Scheduling Algorithm

Input: $N_{\min}, N_{\max}, \Delta,$

Output: workload scheduling decision

Randomly initialize critic network $Q(s, a|\theta^{Q})$ and actor $\mu(s, a|\theta^{\mu})$ with weights θ^{Q} and θ^{μ}.

Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^{Q}, \theta^{\mu'} \leftarrow \theta^{\mu}$

Initialize replay buffer R

For episode = 1 to M **do**

  Initialize a random process N for action exploration

  Receive initial observation new state s_1

  For t = 1 to T **do**

    Select action $a_t = \mu(s_t|\theta^{\mu}) + N_t$ according to the current policy and exploration noise

    Execute action a_t and observe reward r_t and observe new state s_{t+1}

    Store transition (s_t, a_t, r_t, s_{t+1}) in R

    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

    Set $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1}\middle|\theta^{\mu'}\right)\middle|\theta^{Q'}\right)$

    Update critic by minimizing the loss:

    $$L = \frac{1}{N}\sum\left(y_i - Q\left(s_i, a_i|\theta^{Q}\right)\right)^2$$

    Update the actor policy using the sampled policy gradient:

    $$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q\left(s, a\middle|\theta^{Q}\right)$$
    $$|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$$

    Update the target networks:

    $$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
    $$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

  End for

End for

specific to Edge Computing scenarios. Based on CloudSim [16], EdgeCloudSim adds considerable functionality such as network modeling, load generation as well as workload orchestrator, so that it can accurately simulate the real edge computing environment [21].

In the experiment, we adopt edge computing architecture is a two-tier with orchestrator, which is composed of 1 cloud servers, 14 edge servers and a large number of mobile devices. The cloud server has 4 Virtual Machine (VM) with 100 giga instruction per seconds (GIPS) of CPU power, 32GB of random access memory (RAM) and 1000 GB storage. Each edge server has one host that operates 4 VM with 10 GIPS of processing capacity, 2 GB of RAM and 16GB storage to handle the offloaded tasks. In order to simulate different loads, we initialize the number of mobile devices as $N_{\min} = 200$, and increase 200 each time to reach the maximum $N_{\max} = 2000$.

Each mobile device can run four different applications, namely augmented reality, infotainment, and health monitoring applications. Each application represents different task size, latency tolerance, and task length. The task has random input/output file sizes to upload/download and have random lengths.

Algorithm 3 PPO-based Workload Scheduling Algorithm

Input: $N_{\min}, N_{\max}, \Delta,$

Output: workload scheduling decision

For $i \in \{1, \cdots, N\}$ do

  Run policy π_θ for T timesteps, collecting$\{s_t, a_t, r_t\}$

  Estimate advantages $\hat{\mathrm{A}}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$

  $\pi_{old} \leftarrow \pi_\theta$

  For $j \in \{1, \cdots, M\}$ do

    $J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{\mathrm{A}}_t - \lambda \mathrm{KL}\left[\pi_{old}|\pi_\theta\right]$

    Update θ by a gradient method w.r.t. $J_{PPO}(\theta)$

  End For

  For $j \in \{1, \cdots, B\}$ do

    $L_{BL}(\phi) = -\sum_{t=1}^{T} \left(\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)\right)^2$

    Update ϕ by a gradient method w.r.t.$L_{BL}(\phi)$

  End For

  If $\mathrm{KL}\left[\pi_{old}|\pi_\theta\right] > \beta_{high}\mathrm{KL}_{target}$ **then** $\lambda \leftarrow a\lambda$

  Else if $\mathrm{KL}\left[\pi_{old}|\pi_\theta\right] > \beta_{low}\mathrm{KL}_{target}$ **then** $\lambda \leftarrow \lambda/a$

  End if

End For

According to the attractiveness of user, we divide places into three different levels, which directly affects the dwell time that the user spends in the related places. We set the mean waiting time of type 1, 2 and 3 respectively as 60, 30 and 15 minutes. We also set WAN Propagation Delay as 100ms and Lan Internal Delay as 5ms.We set the related parameters in simulations in Table 1.

Table 1 the related parameters in simulations

Parameters		Values
Cloud Sever		has 4 VM
Each VM in Cloud sever	CPU	100GIPS
	RM	32GB
	Storage	1000GB
Edge Server		has 4 VM
Each VM in Edge Server	CPU	10GIPS
	RM	2GB
	Storage	16GB
Number of Mobile Devices		200 to 2000
The mean waiting time	in type 1 place	60 minutes
	in type 2 place	30 minutes
	in type 3 place	15 minutes
WAN Propagation Delay		100ms
Lan Internal Delay		5ms

Results and analysis

In order to illustrate the effectiveness of our proposed DRL-based workload scheduling approach. We compared it with PPO-based and DDPG-based scheduling approaches in terms of average service time, and average VM utilization, failed tasks rate. The results are shown in Fig. 3.

Figure 3(a) shows the average service time of different approaches. We can see that the average service time for each approach increases with the number of mobile devices. At the beginning, the longest of the average service time is PPO-based approach, which is 0.8 seconds, followed by DDPG-based approach is 0.75 seconds, and the shortest is DQN-based approaches,which is 0.7 seconds. As the number of devices increases, the average service time of the three approaches increases. When the number of devices increases to 2000, the average server of DDPG-based approach is the longest, reaching 3 seconds, followed by PPO-based approach reaching 2.5 seconds, and DQN-based approach is 1.9 seconds. Compared with other approaches, our proposed DQN-based approach achieves the shortest average service time among these three approaches and the curve stays relatively flat during the whole experiments, which shows that our proposed DRL-based approach makes the task distribution more balanced.

Figure 3(b) shows the average VM Utilization of different approaches. At the beginning, the lowest of the average VM utilization is PPO-based approach, and the average VM utilization of each approach increases with the number of devices. When the number of devices increases to 1000, the lowest of the average VM utilization is replaced by DQN-based approach, and stayed there until the end, reaching about 24%.

Figure 3(c) shows the failed tasks rate of different approaches. As can be seen from the figure, when the number of mobile devices changes from 200 to 1600, the failed tasks rate of all approaches can be kept below 1% and relatively stable. When the number of mobile devices changes from 1400 to 2000, the failed tasks rate of each approach increases with the number of mobile devices. Among them, the PPO-based approach has the fastest growing, and reaches 6% at 2000 devices, followed by DDPG-based approach, reaching 4%. Compared with other approaches, our proposed DQN-based approach achieves the lowest failed task rate of only 3%.

Although we can see the failed tasks rate of different approaches from Fig. 3(c), we don't know the reasons of task failure from figure. Next, we will analyze the reasons for task failure. In our simulations, Task failure is due to the following reasons:

1)The lack of VM capacity. If the VM utilization is too high, the server does not have enough VM capacity to

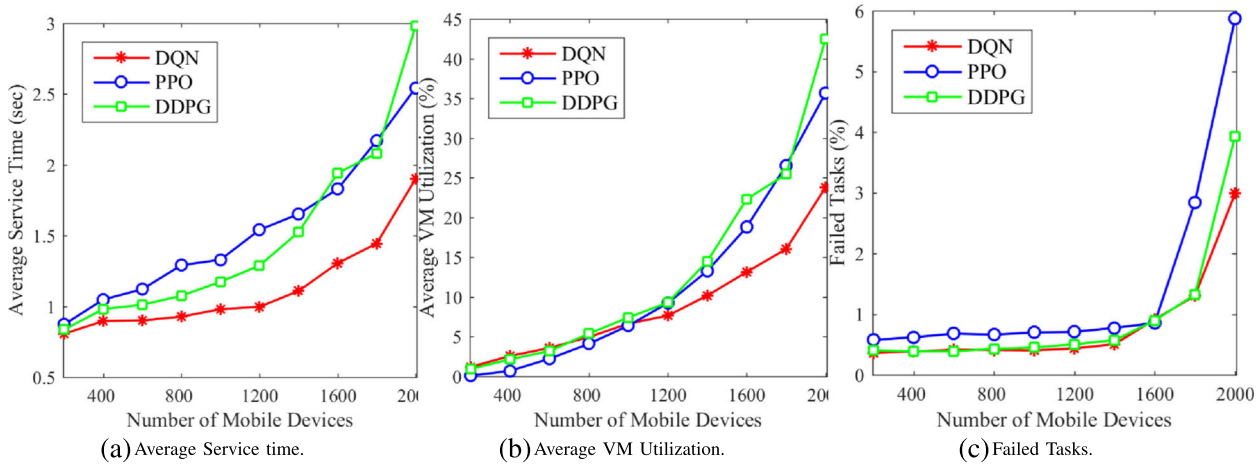

Fig. 3 Comparison of the three aspects: **a** Average Service Time. **b** Average VM Utilization. **c** Failed Tasks

meet new coming tasks, which make task waiting time too long to failure.

2) The mobility of users. If the user leaves his location before getting the response of the previously requested tasks, the tasks will fail because the user is out of the previous WLAN coverage and cannot receive the response from servers.

3) The lack of network resource. If a large number of users simultaneously use the same network resources (such as WLAN, MAN or WAN), it will cause insufficient network bandwidth or even network congestion, resulting in packets loss, and eventually lead to task failure.

The reasons for tasks failure are shown in Fig. 4. It can be clearly seen that, In the PPO-based approach, the main reason for task failure is due to lack of VM capacity. When the number of users reaches 1600, the mobility of users becomes the main reason for task failure. In the DDPG-based approach, the main reason for task failure is due to lack of WAN resource. When the number of users reaches 800, Lack of VM capacity becomes the main reason of task failure In our DQN-based approach, the main reason for task failure is due to MAN failure, followed by mobility of users. In all approaches, there is almost no transmission failure in the WLAN, so the reason of the tasks failure due to the WLAN can be negligible.

Given the above observation, our proposed DQN-based workload Scheduling approach has the best performance in aspects of service time, and VM utilization, failed tasks rate among these three approaches. Especially, When the number of users is large (above 1600), our proposed DQN-based approach still can keep the lowest task failure rate, which shows that the task allocation is more reasonable and effective, and ensures the stability of the system

Convergence analysis

In this section, we will analyze the convergence of our proposed DQN-based algorithm by observing the change of training curves in the training process. Figure 5 presents the training curve of average reward under the DQN-based algorithm during 400 training episodes. It can be seen that the curve always fluctuate upward and downward at the first of 150 training episodes, because the agent is in the exploration stage and the reward value is unstable. As the number of training episodes increases, the agent is changed from the exploration phase enters the learning phase using experience, the algorithm begins to converge and the reward value tends to stabilize.

Conclusion

In this paper, we have investigated workload scheduling in edge computing, aiming at balance the workload, reduce the service time and minimize failed task rate. Considering that edge computing system is a very dynamic environment and affected by several factors. We analyze system model, which includes task model, network model, computational model, according to the Multi-tier edge computing architecture, and formulate the workload scheduling problem based on system model as a hard and online problem. To deal with the challenges of workload scheduling problem, we proposed a DRL-based approach, which can learn from the previous actions and achieve best scheduling in the absence of a mathematical model of the environment. Simulation results show that our proposed approach achieves the best performance in aspects of service time, virtual machine utilization, and failed tasks rate compared with other approaches. Our DRL-based approach can provide an efficient solution to the workload scheduling problem in edge computing.

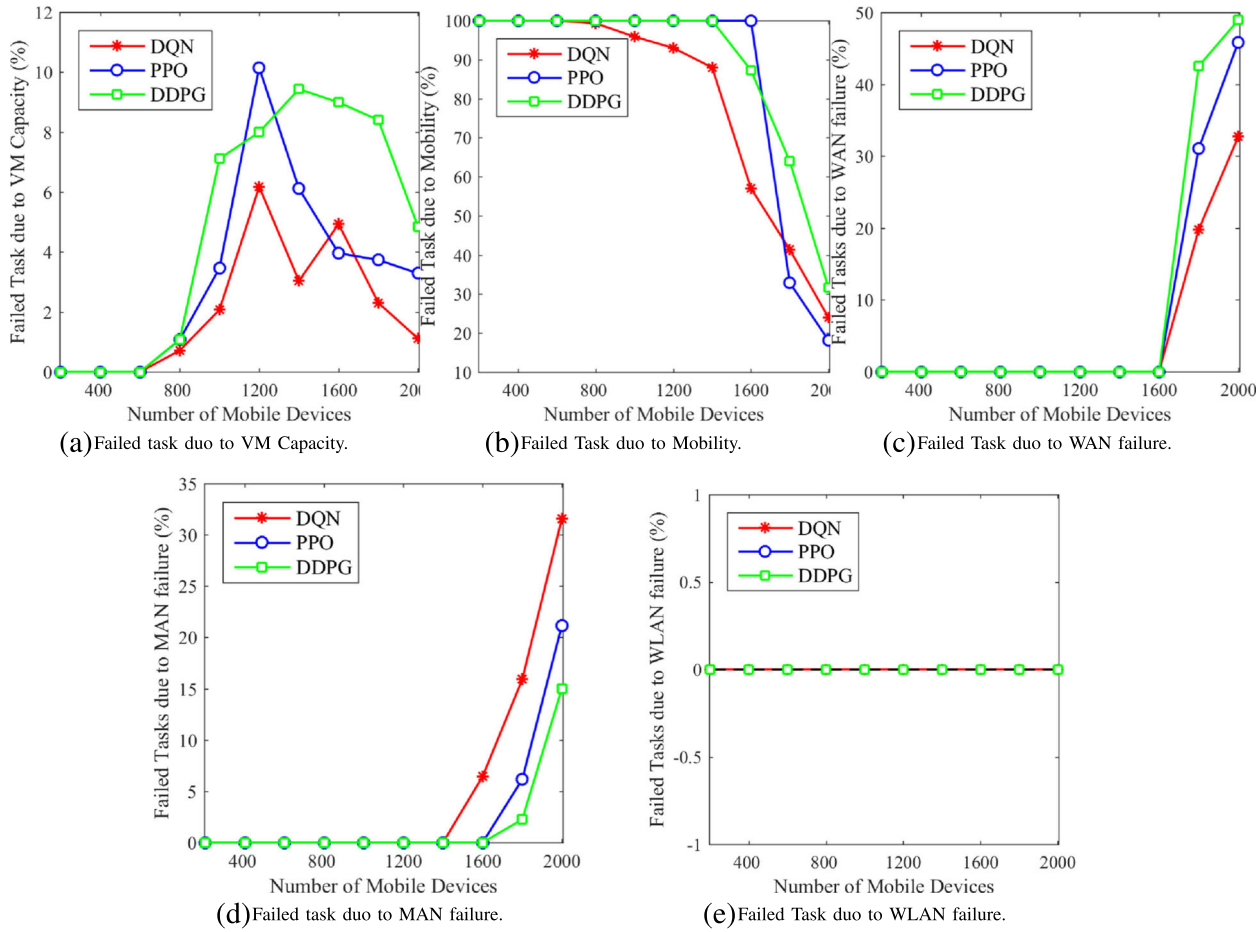

Fig. 4 The Reasons for Task Failure

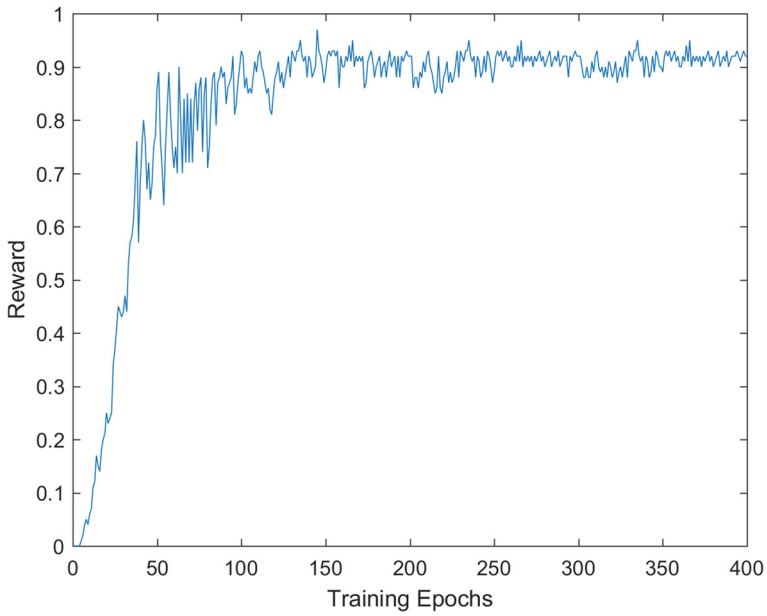

Fig. 5 The Convergence of the proposed DQN-based algorithm

Nomenclature

i	index of a task
∂_i	input data size (in bits) of $task_i$
β_i	output data size (in bits) of $task_i$
c_i	total number of CPU cycles that is required to complete the $task_i$
τ_i	delay constraint of $task_i$
t^i_{com}	communication delay of $task_i$
t^i_{td}	transmission delay of $task_i$
t^i_{up}	upload time of $task_i$
t^i_{down}	download time of $task_i$
t^i_{wait}	waiting time between the uploading to local edge server and starting execution
ω_{wlan}	bandwidth of WLAN
p_n	transmission powers of mobile device
$h_{n,s}$	channel gains between mobile device and edge server
N_0	noise power
f_l	local edge server computing power
f_{nb}	nearby edge server computing power
f_c	cloud computing power
R_{man}	MAN transmission rate
R_{wlan}	WLAN transmission rate
$\lambda_1, \ldots \lambda_{m+1}$	scheduling decision variables
$server_t$	states of all servers at step t
u_n	VM utilization of n_{th} server
$network_t$	states of networks at step t
T_t	total delay of current task

Acknowledgements
This work was supported by the National Natural Science Foundation of China (No.62072146, Learning Driven Edge Intelligent System Performance Optimization Technology), in part by Zhejiang Provincial Key Technology Research and Development Program under Grant No.2019C01059. National Key Technology Research and Development Program of China under Grant No.2019YFB2102100.

Authors' contributions
Methodology: Tao Zheng; Resources: Jian Wan and Jinlin Zhang; Software: Tao Zheng, Jilin Zhang and Congfeng Jiang; Supervision: Jian Wan; Writing original draft: Tao Zheng; Writing review editing: Jian Wan, Jilin Zhang and Congfeng Jiang; All authors read and approved the final manuscript.

Funding
This study was supported by the National Natural Science Foundation of China under Grant No.62072146. Zhejiang Provincial Key Technology Research and Development Program under Grant No.2019C01059. National Key Technology Research and Development Program of China under Grant No.2019YFB2102100.

Availability of data and materials
The data used to support the findings of this study are available from the corresponding author upon request.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 22 March 2021 Accepted: 30 November 2021
Published online: 08 January 2022

References
1. Mao Y, et al. (2017) A Survey on Mobile Edge Computing: The Communication Perspective. IEEE Commun Surv Tutor PP(99):1–1
2. Guo X, et al. (2018) Data offloading and task allocation for cloudlet-assisted ad hoc mobile clouds. Wirel Netw 24(1):79–88
3. Fang W, Li Y, Zhang H, Xiong N, Lai J, Vasilakos AV (2014) On the throughput-energy tradeoff for data transmission between cloud and mobile devices. Inf Sci 283(1):79–93
4. Sanaei Z, et al. (2014) Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges. IEEE Commun Surv Tutor 16(1):369–392
5. Satyanarayanan, Mahadev (2017) The Emergence of Edge Computing. Computer 50(1):30–39
6. Cisco (2018) Cisco global cloud index: Forecast and methodology, 2016-2021. White Paper. pp 1–44
7. Lei L, Xu H, Xiong X, Zheng K, Xiang W, Wang X (2019) Multiuser Resource Control With Deep Reinforcement Learning in IoT Edge Computing. IEEE Internet Things J 6(6):9–10133. https://doi.org/10.1109/JIOT.2019.2935543
8. Sonmez C, Ozgovde A, Ersoy C (2019) Fuzzy Workload Orchestration for Edge Computing. IEEE Trans Netw Serv Manag 16(2):769–782. https://doi.org/10.1109/TNSM.2019.2901346
9. Flores H, Su X, Kostakos V, Ding AY, Nurmi P, Tarkoma S, Hui P, Li Y (2017) Large-scale offloading in the Internet of Things. pp 479–484. https://doi.org/10.1109/PERCOMW.2017.7917610
10. Sonmez C, Ozgovde A, Ersoy C (2017) EdgeCloudSim: An environment for performance evaluation of Edge Computing systems. In: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC). pp 39–44. https://doi.org/10.1109/FMEC.2017.7946405
11. Santoro D, Zozin D, Pizzolli D, De Pellegrini F, Cretti S (2017) Foggy: A Platform for Workload Orchestration in a Fog Computing Environment. In: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). pp 231–234. https://doi.org/10.1109/CloudCom.2017.62
12. Anas A, Sharma M, Abozariba R, Asaduzzaman M, Benkhelifa E, Patwary MN (2017) Autonomous Workload Balancing in Cloud Federation Environments with Different Access Restrictions. In: 2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS). pp 636–641. https://doi.org/10.1109/MASS.2017.68
13. Ma X, Zhou A, Zhang S, Wang S (2020) Cooperative Service Caching and Workload Scheduling in Mobile Edge Computing. In: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications. pp 2076–2085. https://doi.org/10.1109/INFOCOM41043.2020.9155455
14. Nascimento A, Olimpio V, Silva V, Paes A, de Oliveira D (2019) A Reinforcement Learning Scheduling Strategy for Parallel Cloud-Based Workflows. In: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp 817–824. https://doi.org/10.1109/IPDPSW.2019.00134
15. Liu Z, Zhang H, Rao B, Wang L (2018) A Reinforcement Learning Based Resource Management Approach for Time-critical Workloads in Distributed Computing Environment. In: 2018 IEEE International Conference on Big Data (Big Data). pp 252–261. https://doi.org/10.1109/BigData.2018.8622393
16. Hao Y, et al. (2020) Deep Reinforcement Learning for Edge Service Placement in Softwarized Industrial Cyber-Physical System. IEEE Trans Ind Inf PP(99):1–1
17. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence, Vol 30, No 1
18. Qiu C, et al. (2019) Deep Deterministic Policy Gradient (DDPG)-Based Energy Harvesting Wireless Communications. IEEE Internet Things J 6(5):8577–8588
19. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347
20. Liu L, Mitra U (2020) On Sampled Reinforcement Learning in Wireless Networks: Exploitation of Policy Structures. IEEE Trans Commun PP(99):1–1
21. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R (2011) CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp 41(1):23–50

Publisher's Note