Computer Communications 175 (2021) 1–12

Contents lists available at ScienceDirect

Computer Communications

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

Deep reinforcement learning for computation offloading in mobile edge computing environment

Miaojiang Chen [a], Tian Wang [b], Shaobo Zhang [c], Anfeng Liu [a,*]

[a] School of Computer Science and Engineering, Central South University, ChangSha 410083, China
[b] College of Computer Science and Technology, Huaqiao University, Xiamen, 361021, China
[c] School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, 411201, China

ARTICLE INFO

Keywords:
Internet of things (IoT)
Reinforcement learning
Markov decision process
Computation offloading
Deep learning
Mobile edge computing

ABSTRACT

Recently, in order to distribute computing, networking resources, services, near terminals, mobile fog is gradually becoming the mobile edge computing (MEC) paradigm. In a mobile fog environment, the quality of service affected by offloading speeds and the fog processing, however the traditional fog method to solve the problem of computation resources allocation is difficult because of the complex network states distribution environment (that is, F-AP states, AP states, mobile device states and code block states). In this paper, to improve the fog resource provisioning performance of mobile devices, the learning-based mobile fog scheme with deep deterministic policy gradient (DDPG) algorithm is proposed. An offloading block pulsating discrete event system is modeled as a Markov Decision Processes (MDPs), which can realize the offloading computing without knowing the transition probabilities among different network states. Furthermore, the DDPG algorithm is used to solve the issue of state spaces explosion and learn an optimal offloading policy on distributed mobile fog computing. The simulation results show that our proposed scheme achieves 20%, 37%, 46% improvement on related performance compared with the policy gradient (PG), deterministic policy gradient (DPG) and actor–critic (AC) methods. Besides, compared with the traditional fog provisioning scheme, our scheme shows better cost performance of fog resource provisioning under different locations number and different task arrival rates.

1. Introduction

With the popularity of mobile smart devices, such as phones and ipads, many mobile applications have emerged and become popular. For instance, online games and virtual and augmented reality [1–4]. Nevertheless, mobile devices are usually constrained due to resource shortage, such as the CPU computation power and the memory size of servers are limited [5–7]. Furthermore, the quality-of-experience (QoE) and the performance of computation-intensive applications are serious implications by limited computing power when performed at the mobile devices. The resource shortage of computation-intensive applications emerged as a bottleneck for gratifying the quality-of-service (QoS) and the QoE and promotes the generation of edge computing [8,9].

Cloud computing has developed rapidly in recent years, the IoT network computing technology has entered a new era [10,11]. In the industry, many internet technology companies, such as Google, Microsoft, Amazon, Baidu, Alibaba, etc. are developing cloud computing model as a utility. Meanwhile, the above companies have enabled cloud-based services including software as a service (SaaS), platform as a service (PaaS), infrastructure as a service (IaaS) to deal with educational and numerous enterprise-related problems simultaneously. Nevertheless, the great majority of the cloud data-centers are geographically centralized and located far from end-users. Consequently, latency-sensitive and real-time data transmission service requests to be responded by the far-off cloud data centers usually endure network congestion, large round-trip delay. To solve the above problems, edge computing was proposed in [12].

For edge computing, the goal is to make computing devices closer to data sources. To be more precise, edge computing can process data on the edge network [13]. As we know, the edge network is mainly composed of terminals (such as smart phones), edge devices (such as wireless access points, routers, base stations), edge servers and so on. These components can have the essential abilities to support edge computing. Compared with the cloud data centers, edge computing can provide a faster response to computing service requests. However, edge computing does not associate IaaS, PaaS, SaaS and other cloud-based services spontaneously and concentrate more towards the end devices side [14]. Considering the concepts of edge and cloud computing, several computing paradigms, such as mobile edge computing

* Corresponding author.
E-mail addresses: miaojiangchen@csu.edn.com (M. Chen), wangtian@hqu.edu.cn (T. Wang), shaobozhang@hnust.edu.cn (S. Zhang), afengliu@mail.csu.edu.cn (A. Liu).

https://doi.org/10.1016/j.comcom.2021.04.028
Received 4 September 2020; Received in revised form 28 December 2020; Accepted 25 April 2021
Available online 30 April 2021

(MEC) [15,16] and mobile cloud computing (MCC) [17–19] are proposed as potential extensions of cloud and edge computing. As an edge-centered computing model, MEC has attracted wide attention. MEC is considered as one of the key promoters of the development of modern cellular base stations. It offers edge servers and cellular base stations to be operated combinedly [20].

As mentioned in the Cisco Cloud Index (2013–2018), although most internet traffic has been initiated or terminated in data centers since 2008, nearly two-thirds of the tasks in traditional IT space were performed in the cloud in 2016. Nevertheless, cloud computing faces various challenges, such as network bandwidth, security, reliability and latency [21]. In order to deal with these problems, Bonomi et al. [22] proposed a new paradigm, named fog computing in 2012. Meanwhile, fog computing is a horizontal architecture to distribute, storage, networking resources, control and computing close to end-user devices as possible, while sending onerous tasks to the remote cloud when necessary [23,24].

As we know, mobile cloud computing is gradually turning into a common paradigm for mobile terminals [25,26], such as social networking, video streaming, mobile health care services, mobile video games and education [27]. With the explosive growth of mobile data traffic, mobile service providers deliver smooth services by using the mobile cloud.

Amini et al. [28] proposed to expedite the collaboration of different parts to develop and manage sustainable and smart cities containing smart technologies. Meanwhile, latency and reliability-aware task offloading and resource allocation for mobile edge computing is formulated as computation and transmit power minimization [29]. Fog and edge computing paradigm are regarded as capable of satisfying the resource requirements for the emerging mobile applications. However, motivated by the success of deep learning (DL) in a wide spectrum of fields, it is promising that using reinforcement learning to solve the mobile fog resource allocation problem.

In 2019, Sun et al. [30] proposed a deep reinforcement learning-based joint mode selection and resource management approach to solve the resource management in Fog radio access networks (F-RANs). Wang et al. [31] proposed a dynamic reinforcement learning algorithm to solve the problem of traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications problems. Wei et al. [32] proposed using a natural actor–critic deep reinforcement learning to solve the joint optimization of caching, computing, and radio resources for fog-enabled IoT. In summary, mobile fog is a complementary model for fog computing [33].

Although good research results have been achieved, there are some deficiencies in this research field. Such as, the dynamicity of the mobile environment lead to the network state spaces is enormous and difficult to deal with it. Meanwhile, the research of [30–32] all use the DQN methods to learn the optimal stochastic policy, however, DQN cannot overcome the disadvantage of overestimation, which caused the result that the estimated value is larger than the real value.

Therefore, a DDPG reinforcement learning scheme to model a dynamic environment appropriately is considered. The main contributions in our works as follows.

(1) We modeled the offloading blocks (include code block ID, application ID and method ID) and network node states discrete event system model as a Markov decision process (MDP), which can realize the offloading computing without knowing the transition probabilities among different network states. Compared with traditional fog offloading methods, using Markov decision processes, current offloading state s_t contains all history-related information, the system only needs to consider the current state to make the next decision with ignoring the influence of time.

(2) In the mobile fog environment, the code offloaders have many states and offloading actions and traditional fog offloading methods are hard to deal with in this complex environment. The DDPG method is used to solve the problem of state space explosion and trained to predict resource allocation action to find an optimal decision policy. Furthermore, we performed group dissolution with the DDPG algorithm to avoid the agents select ill to compete and solve the problem of overestimation.

(3) According to experiment theoretical analysis, the results show that our proposed learning-based scheme achieves 20%, 37% and 46% performance improvement compared with PG, DPG and AC methods. Furthermore, for the system cost of fog resource provisioning, we proposed scheme that has a better performance compared with the traditional fog resource provisioning scheme.

The rest of the paper is organized as follows. In Section 2, we described the mobile fog model with the reinforcement learning method. In Section 3, we formulate the problem of designing an optimal policy in different network environment to offload code blocks. We detail the deep deterministic policy gradient algorithms for stochastic computation in fog network. In order to verify the effectiveness of the algorithm, Section 4 shows the result of the numerical experiments. Finally, we make the conclusions in Section 5.

2. Preliminaries and background

As a distributed computing model, fog computing serves between Internet of Things devices/sensors and cloud data centers.

Mobile cloud computing is an offloading platform of mobile stations to ensure reliable services toward mobile devices. As we know, the wireless-fidelity (Wi-Fi) and the hierarchical architecture of long term evolution (LTE) is used in mobile fog computing. The mobile fog is established at the edge of the network in the architecture. In mobile fog computing, we use the access point controller (APC) and the access point (AP) as the fog nodes.

2.1. Mobile device network

In mobile wireless networks, mobile devices deliver information by interconnecting various services and applications. The mobile fog architecture is showed in Fig. 1. In this architecture, the proposed mobile fog model can satisfy the demands for considerable computational and communication capacity. As shown in Fig. 1, cloud computing has an obvious disadvantage, that is, the communication distance from the mobile device to the remote cloud server is too long, resulting in high communication delay. Therefore, the mobile cloud is not enough to meet various wireless network applications with delay constraints and high computing power.

In this paper, we discuss the algorithm of offloading computation with reinforcement learning in mobile fog. Using mobile agents, we can offload code blocks from mobile stations with resource shortage to mobile fog by using basic block migration policy. Inspired by [34,35], the code offloading framework can be constructed as Fig. 2. In Fig. 2, the compiler translates programming language to machine language. The front-end executes semantic and syntax analysis and the back end produces byte code, grouping basic blocks. The code offloader is modeled as an N-queen problem with six code blocks. The register values of code 1 are occupied by code 2 and code 3, and the register values of code 2 are occupied by code 6. The above code blocks are divided into five compatible sets, that is, they are completely independent and can be executed in parallel.

For each basic block, the manager of the application assigns block ID, method ID and application ID. In addition, when the application is running, the code manager storage the CPU execution and utilization information. The code blocks will be offloading to neighboring mobile Fog nodes. We can realize the availability of mobile fog nodes by learning to offload the code blocks if necessary. If the average memory and CPU execution and availability are less than the available CPU and memory, the manager of the application will execute the basic block. Or else that it will determine the expected execution time of host processing (EET_H) based on the available CPU and memory of

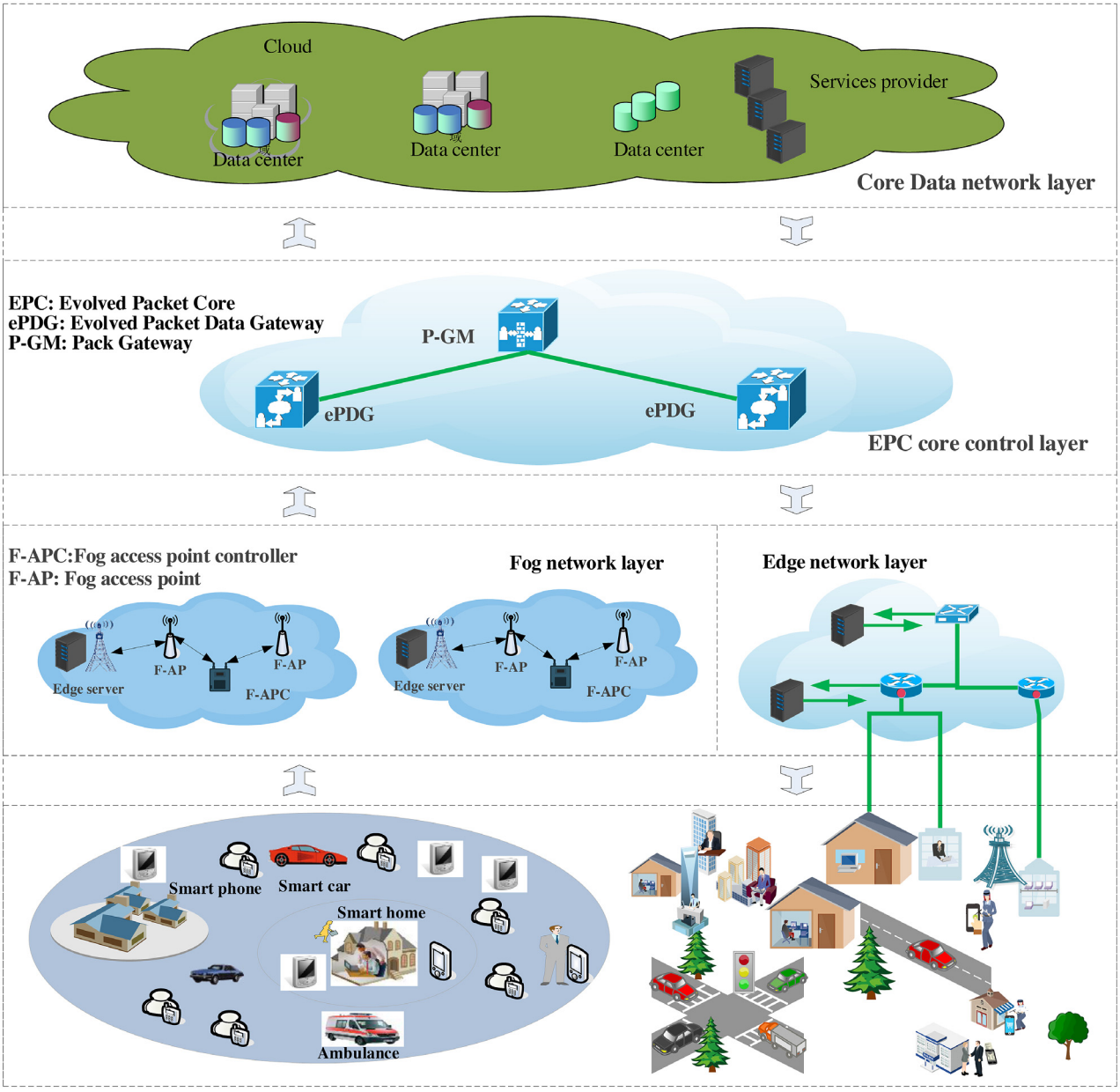

Fig. 1. The mobile fog architecture.

mobile devices. Meanwhile, it also demands F-APC for EET of Fog processing (EET_H) based on available CPU and memory. Finally, compare the time results that is, (EET_F) < (EET_H), the code offloader will make the decision for offloading. Furthermore, the code offloader executes the BFS algorithm and locates the independent blocks. For parallel execution, the code offloader offloaded the independent blocks in mobile Fog.

In order to realize the mobility of both fog nodes and mobile stations, the basic blocks will move in different Fog nodes in neighboring Fog or the same fog. The ePDG node controlled the communication between different mobile fogs. And F-APC nodes are responsible for offloading big data analytics for cyber–physical systems and migration executed to realize the mobility of the mobile stations. In an edge mobile fog, the ePDG node can migrate the basic blocks for load sharing and load balancing. The notations are defined in Table 1.

2.2. Reinforcement learning

Markov Decision Processes (MDPs) provides an idea for modeling decision-making problems, which results are partly random and under the control of an agent. An MDP is denoted as a tuple (S, A, p, r), where S represents a set of states, A is a set of actions, p is a transition probability from state s to s' when executed action a. And r is the reward obtained after action a is executed. π is defined by a "policy", which maps a state to an action. In an MDP, the goal is finding an optimal policy $\pi^* : S \rightarrow A$ for the agent to minimize the cost for the system. Value function $V^\pi : S \rightarrow R$ denotes the expected value gained by π. In Q-learning algorithm, the value function can be expressed as follows:

$$V^\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma r_t(s_t, a_t) | s = s_0] \tag{1}$$

$V^*(s) = max_{a_t}\{E_\pi[r_t(s_t, a_t) + \gamma V^\pi(s_{t+1})]\}$ denotes the optimal value function. Furthermore, we can denote optimal Q -function as $Q^*(s, a) \triangleq r_t(s_t, a_t) + \gamma E_\pi[V^\pi(s_{t+1})]$. Then, we have $V^*(s) = max_a Q^*(s, a)$. The Q function can be updated as:

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha_t[r_t(s, a) + \gamma \max_{a'} Q_t(s, a') - Q_t(s, a)] \tag{2}$$

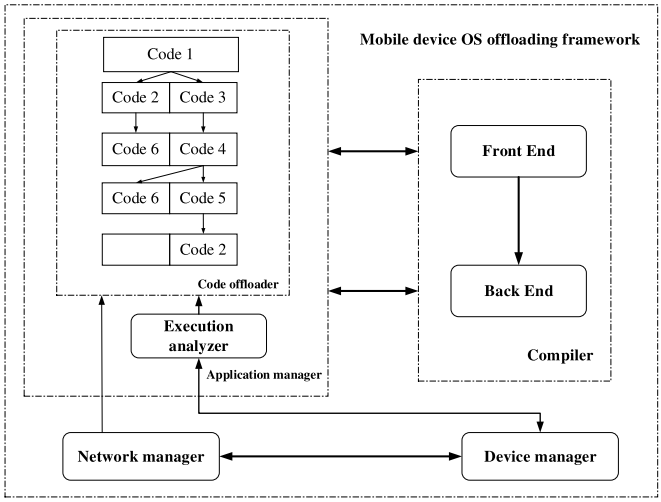

Fig. 2. The code offloading framework.

where, α_t is learning rate. However, the updated rule must satisfy convergence theorem [36] to ensure convergence.

When S and A are small enough, the Q -learning can obtain an optimal policy π^*. Nevertheless, with complicated systems, these spaces and actions are large in practice. Therefore, the Q-learning approach unsuited to search for the optimal policy. As a result, the Deep Q-Learning (DQL) algorithm is presented to solve this problem [25].

As mentioned in [37], the reinforcement learning algorithms with nonlinear function approximator may not be stable to obtain the average reward. To address this problem, the experience replay method is used in DQN. By using ϵ -greedy policy, experiences are generated randomly. Next step, random samples are selected by DQN, that is, mini-batches of transitions from the experience replay D to train the deep neural network. Through training deep neural network, the Q-value is obtained and will be obtained new experiences. Meanwhile, these experiences will be stored in D. By using old and new experiences, the experience replay mechanism enables the deep neural network trained more efficiently. Furthermore, the transitions are more identically distributed and independent by using the experience replay.

The value-state function Q^* of DQN model with the loss function updated written as follows:

$$\Gamma(\theta) = E_{s,a,r,s'}[(Q^*(s,a|\theta) - y)^2] \tag{3}$$

where $y = r + \gamma max\overline{Q}^*(s', a'))$. $\overline{Q}$ is a target Q function.

The idea of using deep network and experience pools in DQN has been applied in deep deterministic policy gradient (DDPG) [38]. DDPG can be seen as a combination of deep and deterministic policy gradient. Deep is a deeper network structure. The policy gradient [39] is a strategy gradient algorithm, which can randomly select actions according to the learned strategies (action distribution) in continuous action space. The purpose of deterministic is to limit policy gradient to perform a random selection and only output one action. In the policy gradient algorithm, for maximizing the $G(\theta) = E_{s\sim p^\pi, a\sim\pi_\theta}[R]$ by executed the step $\nabla G(\theta)$, and $R_t = \sum_{i=t}\gamma^{i-t}r(s_i, a_i)$. The optimal strategy is determined by the probability distribution function, and the optimal action in the current state is obtained at each step according to the probability distribution. The random strategy is adopted to generate the action $a_t \sim \pi_\theta(s_t|\theta^\pi)$. Hence, we can write the objective function as follows:

$$G(\theta) = \int_S p^\pi(s)\int_A \pi_\theta(s,a)r(s,a)dads = E_{s\sim p^\pi, a\sim\pi_\theta}[R] \tag{4}$$

the gradient of policy is defined as follow:

$$\nabla G(\theta) = E_{s\sim p^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s)Q^\pi(s,a)] \tag{5}$$

Table 1
Notation definitions.

Notations	Description
r	Reward
a	Action
s	State
$Q(s,a)$	Action value function
$V(s)$	State value function
π	Policy
p	State transition probability
γ	Discount factor
α	Learning rate
θ	The weight of deep network
$\Gamma(\theta)$	Loss function
$G(\theta)$	Reward expectation function
$\nabla G(\theta)$	The gradient of reward expectation function
Q' net	Target network
F-AP	Fog access point
F-APC	Fog access point controller
τ	The service time
P_M	The steady-state distribution of searching M blocks
N_b	The expected number of blocks
$E_{i,j}$	The expect response time
ϕ	The risk factor
R_{req}	The required response time of the service level agreement
I	The performance index
δ	The cooperation factor

Because the policy gradient is a random policy, it is necessary to sample the probability distribution of the optimal policy in order to obtain the current action, and integral the whole action space in each step of the iteration process, so the computation amount is large. In order to solve this problem, [40] presented a novel algorithm named deterministic policy gradient (DPG), which extended the policy gradient algorithm. In DPG algorithm, an action is determined directly by the function μ_θ, that is, $a_t = \mu(s_t|\mu^\mu)$. The performance objective function can be written as follows:

$$G(\mu_\theta) = \int_S p^\mu(s)r(s,\mu_\theta(s))ds = E_{s\sim p^\mu}[r(s,\mu_\theta(s))] \tag{6}$$

the gradient of DPG is written as follows:

$$\nabla_\theta G(\mu_\theta) = E_{s\sim p^\mu}[\nabla_\theta\mu_\theta(s)\nabla_a Q^\mu(s,a)|a=\mu_\theta(s)] \tag{7}$$

The deep deterministic policy gradient (DDPG) is a variant algorithm based on DPG with deterministic policy and DQN. Combining the above two algorithms, DDPG is adopted the deterministic policy μ_θ to select action $a_t = \mu(s_t|\theta^\mu)$. Where μ_θ is a parameter of the policy network that generates deterministic action.

Inspiring by the actor–critic and PG algorithms, DDPG uses the Q^μ as actor, $Q(s,a)$ as critic. Therefore, the objective function of DDPG can be defined as follow:

$$G(\theta^\mu) = E_{\theta^\mu}[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots] \tag{8}$$

$Q(s,a)$ is expressed as the expected reward of selecting an action under the deterministic policy μ_θ. Using DQN and Q network to approximate $Q(s,a)$. Since in continuous space, the expected reward can be obtained by integral.

$$G_\beta(\mu) = \int_S P^\beta(s)Q^\mu(s,\mu(s))ds = E_{s\sim p^\beta}[Q^\mu(s,\mu(s))] \tag{9}$$

where β is behavior policy. In the common reinforcement learning training process, ϵ-greedy strategy is used to balance exploration and exploit. Similarly, for DDPG algorithm, Uhlenbeck–Ornstein (UO) stochastic processes are used as the random noise. UO processes have well correlated in sequences, which enables agents to explore environment exploration with momentum attributes in order to explore potential better strategies. Hence, we introduce random noise into the decision-making mechanism of action in the training process. The stochastic process is shown in Fig. 3.

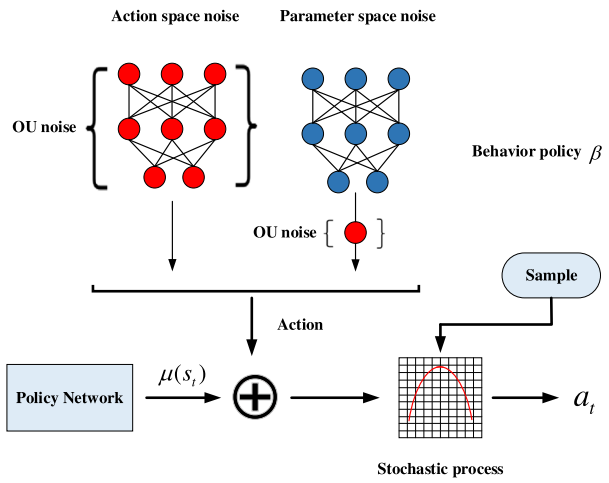

Fig. 3. Random noise into the decision-making mechanism of action in the training process.

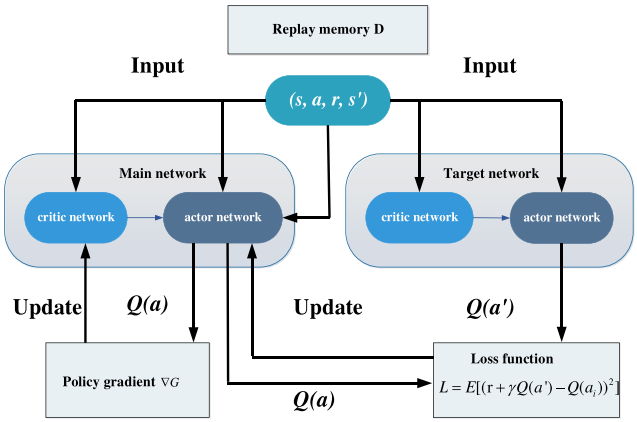

Fig. 4. The DDPG network structure.

Silver [28] proved that the gradient of the objective function using the policy μ is equivalent to the gradient of the Q function using the policy μ, that is,

$$\frac{\partial G(\theta^{\mu})}{\partial \theta^{\mu}} = E_S[\frac{\partial Q(s,a|\theta^{Q})}{\partial \theta^{\mu}}] \tag{10}$$

For deterministic policy, $a = \mu(s|\theta^{\mu})$. Hence, the gradient of actor network can be written as follow.

$$\frac{\partial G(\theta^{\mu})}{\partial \theta^{\mu}} = E_S[\frac{\partial Q(s,a|\theta^{Q})}{\partial a}\frac{\partial \pi(s|\theta^{\mu})}{\partial \theta^{\mu}}] \tag{11}$$

Furthermore, we have:

$$\begin{aligned}\nabla_{\theta} G_{\beta}(\mu_{\theta}) &= \int_S p^{\beta}(s)\nabla_{\theta}\mu_{\theta}(s)Q^{\mu}(s,a)ds \\ &= E_{s\sim p^{\beta}}[\nabla_{\theta}\mu_{\theta}(s)Q^{\mu}(s,a)]\end{aligned} \tag{12}$$

On the other hand, the gradient of critic network can be written as follow.

$$\frac{\partial L(\theta^{Q})}{\partial \theta^{Q}} = E_{s,a,r,s'\sim D}[TargetQ - Q(s,a|\theta^{Q})\frac{\partial Q(s,a|\theta^{Q})}{\partial \theta^{Q}}] \tag{13}$$

where $TargetQ = r + \gamma Q'(s', \pi(s'|\theta^{\mu'})|\theta^{Q'})$.

The DDPG network structure is shown in Fig. 4.

From Fig. 4, we can see that there are two networks: the main network and the target network. The two networks are two layers of fully linked neural networks. The actor outputs a specific action, while the critical network outputs a specific Q value. First, the agent takes a batch of data from the experience pool D (replay memory), and then trains actor network and critic network. Second, the action network selects an action a_t according to behavior policy μ. The environment executed a_t and return the reward r_t and the next state s_{t+1}. The critic network updates the parameters according to the loss function.

3. System model

Resource requests of various applications and mobile stations are dynamic, which urges us to apply the RL algorithm to the resource provisioning [41] of mobile fog. Inspiring by the distributed performance of mobile fog, we apply the distributed reinforcement learning to solve the code offloading issue of mobile fog. In this paper, there are three different options to offload codes: (1) mobile fog in neighboring mobile stations O_1. (2) the near mobile fog to deal with load balancing and mobility issues, O_2. (3) Manage computing requirements and huge traffic in the remote public cloud, O_3. By using multi-mobile agents to search the optimal options. The agents of mobile fog will learn from the environment and experience by dissolution and formation of groups (DFG) algorithm [42,43], that is, the DPG algorithm will be used to learn in the multi-agent system and select actions.

The action set A have ten cases in mobile fog:

(1) if offload in O_1 is a_1.
(2) if offload in O_2 is a_2.
(3) if offload in O_3 is a_3
(4) if from O_1 migrate to O_2 is a_4.
(5) if from O_2 migrate to O_1 is a_5.
(6) if from O_1 migrate to O_3 is a_6.
(7) if from O_3 migrate to O_1 is a_7.
(8) if from O_2 migrate to O_3 is a_8.
(9) if from O_3 migrate to O_2 is a_9.
(10) if only migration within O_1 is a_{10}.

Each agent s_i will be in charge of a specific action in the mobile fog system. According to the Dissolution and Formation of Groups (DFG) algorithm, we compete with agents in groups. A group can contain multiple agents or a single agent. Each of the groups has a leader to participate in the competition. For instance, certain agents intend to move a basic block into a fog node, other agents intend to offload it in a neighboring fog node, however, two agents must compete with each other with their capability of dealing with this task. Meanwhile, one independent block can be performed in parallel to reduce time cost and one dependent block must be executed in sequential order. Hence, if concurrent execution cannot use in mobile applications code, actions may become incompatible.

Furthermore, we defined state S as networking bandwidth capability, processing and storage of Fog nodes, that is, $S = s_1, s_2, \ldots, s_n$, where $s_i = (bw_i, cpu_i, storage_i)$. In this paper, no matter what kind of network environments, no agent can obtain global state knowledge (that is, states of all mobile fogs). Each agent only knows the local state, but agents can collaborate and communicate with each other to offload the code in the optimal fog node.

As mentioned above, an agent is responsible for an action. For instance, considering the agents of O_1 as a group, an agent placement of basic block in F-APC, another agent placement of in $F - AP_2$. The estimated capability of group member agents is $E_{i,j}$, the group leader takes a competition bid $C_{i,j}$ to arrange compatible basic blocks in O_1

$$C_{i,j} = \phi * E_{i,j} + \omega * E_{i,j} \tag{14}$$

where, i, j represent two agents, respectively. ϕ is a constant factor of risk estimation. ω is a noise factor to avoid falling into local minimum. Considering the response time of performing the block as the estimated capability. In order to figure out the estimated response time, we analyze the queue states of fog nodes by queuing theory. We assume each mobile fog nodes have a single server, M represents a server that

can process the maximum number of blocks. ξ is the arrival rate. τ is service time. According to [44], the expected response time in O_1 is:

$$E[T_{O_1}] = \frac{N_b}{\xi(1-P_M)} \tag{15}$$

where, P_M is the steady-state distribution of searching M blocks. When the fog node is busy, the probability is p. N_b is the expected number of blocks, which is defined as follows:

$$N_b = \frac{\frac{\xi}{\tau}(1-(M+1)(\frac{\xi}{\tau})^M + M*(\frac{\xi}{\tau})^{M+1})}{(1-\frac{\xi}{\tau})(1-(\frac{\xi}{\tau})^{K+1})} \tag{16}$$

If a group leader wants to move a block in O_2, that is, in the distant mobile fog, the expected response time in O_2 is:

$$E[T_{O_2}] = \frac{Q_l + p(1-P_M)}{\xi(1-P_M)} + CL_{1,2} \tag{17}$$

where Q_l is the expected queue length in O_2. $CL_{1,2}$ denotes the communication latency between O_1 and O_2. Furthermore, if the leader wants to move a code block in O_3 (the public cloud), the expected response time in O_3 can be defined as:

$$E[T_{O_3}] = \frac{1}{\tau} + \frac{1}{\tau(N_c-\sigma)} \sum_{l=n}^{\infty} P_0 \frac{\sigma^l}{N_c!((N_c)^{m-N_c})} + CL_{1,3} \tag{18}$$

where, N_c is the number of cloud servers. P_0 is the probability of zero blocks. σ is the utilization factor. Similarly, $CL_{1,3}$ denotes the communication latency between O_1 and O_3.

And estimated capability $E_{i,j}$ is defines as the response time can be written as follows:

$$E_{i,j} = \begin{cases} \dfrac{N_b}{\xi\left(1-P_M\right)}, \\ \quad \text{if } i = \{1,5,7,10\}, \text{ where } a_i \in A \text{ and } s_j \in S \\ \dfrac{Q_l + p\left(1-P_M\right)}{\xi\left(1-P_M\right)} + CL_{1,2}, \\ \quad \text{if } i = \{2,4,9\}, \text{ where } a_i \in A \text{ and } s_j \in S \\ \dfrac{1}{\tau} + \dfrac{1}{\tau\left(N_c-\sigma\right)} \displaystyle\sum_{l=n}^{\infty} P_0 \frac{\sigma^l}{N_c!((N_c)^{m-N_c})} + CL_{1,3}, \\ \quad \text{if } i = \{3,6,8\}, \text{ where } a_i \in A \text{ and } s_j \in S \end{cases} \tag{19}$$

According to Eq. (14), using $E_{i,j}$ different agent group: {1, 5, 7, 10}, {2, 4, 9}, {3, 6, 8} to determine the competition bid $C_{i,j}$. The group that has maximum competing value will be the winner, executes the action.

$$C_{i,j} = \underset{k}{argmax}\{C_{k,j}\} \tag{20}$$

The policy of credit assignment is to present the reward value to the former winner agent to set the environment. Assigning credit to the predecessor by using the bucket-brigade (BB) model. Assume the agent i is a current winner at state s_j, agent k at state s_l is the immediate predecessor of agent i. According to the BB model, i reduces its $E_{i,j}$ in formula (20).

$$E_{i,j} = E_{i,j} - \phi * E_{i,j} + R_e \tag{21}$$

where R_e is the external reward, ϕ is the risk factor. The external reward R_e can be defined as follow.

$$R_e = \begin{cases} w * \dfrac{E_{i,j} - R_{req}}{E_{i,j}}, \text{if } E_{i,j} \le R_{req} \\ w * \dfrac{E_{i,j}}{R_{req}}, \text{ if } E_{i,j} > R_{req} \end{cases} \tag{22}$$

where, w is the reward weight factor, R_{req} is the required response time of the service level agreement. Furthermore, k receives credit for i by formula (22).

$$E_{k,l} = E_{k,l} + \phi * E_{i,j} \tag{23}$$

Algorithm 1 Deep Deterministic Policy Gradient

1. **Input:**
2. actor $\mu(s,\theta^{\mu})$, critic network $Q(s,a|\theta^{Q})$
3. target network Q'
4. $\theta^{Q'} \leftarrow \theta^{Q}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$, respectively.
5. replay memory D.
6. **Output:**
7. Optimal policy π
8. **while** $episode \le M$ **do**
9. For action exploration to initialize a random process N
10. Receive observation state s_1
11. **while** $i \le T$ **do**
12. According to current policy to select action
$$a_i = \mu(s_t|\theta^{\mu}) + N_i$$
13. Exploration noise
14. Perform a_i, observe reward r_i and state s_{i+1}
15. Store transition (s_i, a_i, r_i, s_{i+1}) in replay memory D.
16. From D to sample a random mini-batch of N transition (s_t, a_t, r_t, s_{t+1})
17. $$y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s||t+1|\theta^{\mu'})|\theta^{Q})$$
18. using the minimizing loss to update critic:
$$Loss = \frac{1}{N}\sum_{t}(y_i - Q(s_i, a_i|\theta^{Q}))^2$$
19. using sample policy gradient to update actor policy:
$$\nabla_{\theta^{\mu}} G \approx \frac{1}{N}\sum_{t} \nabla_a Q(s,a|\theta^{Q}) \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s=s_t, a=\mu(s_t)}$$
20. the target network update:
$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$
21. **end while**
22. **end while**

In a mobile fog system, agents all need to responsible for different actions. According to compatible agents grouping, group of agents is not always fixed, the agent can change its group based on its usefulness and performance. In the agent group, the performance index I of last e episodes can be defined as follow.

$$I_{i,j}^{[t+1]} = \frac{1}{e}\sum_{v=t-e+1}^{t} E_{i,j}^{[\mu]} \tag{24}$$

where $(t+1)$ is the actual episode, μ is the last episode, and i, j denotes the agent and state, respectively. Based on the determined gliding mean, the performance of agents will degrade over time by formula (24).

$$I_{i,j}^{[t+1]} \le \delta * E_{i,j}^{[t-e]} \tag{25}$$

where δ is the cooperation factor. As we know, using low-dimensional observations, the deep deterministic policy gradient algorithm can learn competitive policies. The DDPG algorithm is shown in Algorithm 1.

Algorithm 2 The group of agent formation

1. Input: a random cooperate group set G
2. Output: $G \setminus \{G_i, G_k\}$
3. **while** $G \ne \emptyset$ **do**
4. Search G_i of $E_{i,j} = \underset{l}{max} E_{l,j}: G_l \in G$
5. Send cooperation information to other agents.
6. Receive cooperation response information from G_l.
7. Among the compatible G_l agents, select the maximum p agents of G_k as:
$$E_{k,j} = \underset{l}{max} E_{i,j}: G_l \in \text{compatible } G$$
8. Leader of the group: G_i
9. Update $G = G \setminus \{G_i, G_k\}, \forall G_k \in \text{compatible } G$
10. **end while**

The group of agent formation is presented in Algorithm 2. Since incompatible agents will execute incompatible actions, so in the group formation must maintain the compatibility of agents. The procedure of group dissolution is shown in Algorithm 3.

In order to avoid the agents select ill compete, is also necessary to execute group dissolution. As shown in Algorithm 3, if the group gliding means less than the minimum level, the group leader will dissolve this group.

Algorithm 3 The procedure of group dissolution

1. **Input:** all group G_i
2. **Output:** cooperate set G
3. **while** $G_i \neq 0$ **do**
4. Solve the gliding mean:

$$I_{i,j}^{[t+1]} = \frac{1}{e}\sum_{v=t-e+1}^{t} E_{i,j}^{[\mu]}$$

5. If $(I_{i,j}^{[t+1]} < \eta)$
6. Initial E_{init} (estimated response time):

$$I_{i,j}^{[t+1]} \leq \eta * E_{init}$$

7. Free members of the group, set them in a cooperate set.
8. **end while**

Using the reinforcement learning approach, we present an algorithm in the mobile fog to offload the code block. The detailed algorithmic steps as shown in Algorithm 4.

Algorithm 4 The procedure of group dissolution

1. Offload computation by mobile fog nodes (that is, F-APC, F-AP)
2. Read the blocks of mobile code.
3. **while** each set of compatible blocks having P block **do**
4. Agent of fog nodes check memory, CPU, bandwidth
5. Using Algorithm 2 to move an agent group
6. According to formula (19) to estimate $E_{i,j}$
7. According to formulas (14) and (20) to compete with another agents.
8. Using DDPG (Algorithm 1) to select the winner (action) of this episode.
9. The agent executed optimal action accordingly.
10. By external rewards and adjusting estimates to receive credits.
11. Using Algorithm 3 dissolve groups
12. **end while**

Consider one fog node is attached to the cloud gateway (CGW) at Location O_1. In a certain period of time, the mobile device initiates an application at N different locations, which denoted as $O_i, i \in 1, \ldots, N$. At each O_i, some tasks will be sent and further executed in the fog node. We model fog processing and transmission delay as a two-layer queueing system. For the fog processing queue, the tasks of fog can schedule to different virtual machines (VMs). Let p_{ij} represents the probability, which means a task assigned to VM j at O_i. Assume that the tasks are equally distributed to all VMs, that is $p_{ij} = \frac{1}{x_i}, \forall i \in N, j \in 1, \ldots, x_i$, where x_i represents the number of rented VMs in O_i. The load of fog queue can be written as.

$$\xi_i = \frac{\lambda_i l_i v_i}{x_i u}, \forall i \in N \tag{26}$$

where, l_i is the input data size and v_i is the computation intensity of CPU. λ_i is the traffic arrival rate. u is the VM computing capacity.

Furthermore, the fog delay is written as.

$$d_i^c = \frac{l_i v_i}{u(1-\xi_i)}, \forall i \in N. \tag{27}$$

Table 2
DDPG architecture.

Net	Layer	Activation	Units
Critic	Fully connected	REL	50
	As above		50
	As above	Tanh	50
	As above		50
	As above	RELU	50
	As above		50
Actor	Fully connected		50
	As above		25
	As above	Sigmoid	1

For the transmission queue, the r_i at O_i can be defined as:

$$r_i = B_w \log(1 + \frac{p_i G_i}{N_0 B_w}), \forall i \in N. \tag{28}$$

where B_w is the bandwidth, G_i is the channel gain and p_i is denoted transmission power.

The minimize cost of fog resource provisioning and power control can be written as.

$$R_g = C\sum_{i=1}^{N} x_i, s.t. \frac{l_i}{r_i - \lambda_i l_i} + \frac{l_i v_i}{u - \frac{\lambda_i l_i v_i}{x_i}} \leq D_i \tag{29}$$

where D_i is the completion deadline, C is the VM cost.

The energy cost denotes as W_i, W_1 is mobile devices energy cost, W_2 is cloud node energy cost, W_3 is fog node energy cost. W_i is defined as:

$$W_i = \vartheta_i P_i h E_{i,j} \tag{30}$$

where ϑ_i are the energy efficiency factor of mobile devices, cloud and fog. P_i are the transmit power of mobile devices, cloud and fog. h is the channel power gain of devices. $E_{i,j}$ are the estimated response time during offloading.

Next, we given the details about the system complexity. Suppose the deep neural network has L layers, and M is the input layer size. u_i is the neural units number of ith layer. Let $Y = L * u_i + \sum_{i=1}^{l-1} u_i u_{i+1}$. Hence, the computational complexity of the agent is $O(Y)$. In the training process of deep learning, one forward propagation complexity is $O(Y)$. Setting H episodes with T time-steps, and N agents. The computational complexity is $O(YHTN)$. Assumed that convergence is achieved after Z iterations, the total computational complexity is $O(YHTNZ)$. The system model architecture is shown in Fig. 5.

4. Experiment simulations

In this section, we evaluate the proposed mobile fog algorithms by conducting multiple simulations. In our simulation experiment, we will connect two adjacent mobile fogs. Meanwhile, each mobile fog all has three F-AP nodes and one F-APC node. The mobile fog nodes are all connected to the cloud with virtual machines.

For reinforcement learning, we use a graphics processing unit (GPU), which is Nvidia GTX 2080Ti. The CPU i7-9750H. The programming environment is tensorflow-gpu-1.11.1, python-3.6 on Linux system.

4.1. System state

We will take the parametric analysis of DDPG in this section. And show the influence of the parameters on the code offload training process and experiment results. By learning from the experience of previous studies and a lot of experimental tests, we found that using the parameters in Table 2, we can get better performance.

In the DDPG, there are 3 fully connected layers to construct the target subnetwork and the online subnetwork for the actor network. The critic network is generated by double fully connected layers. In

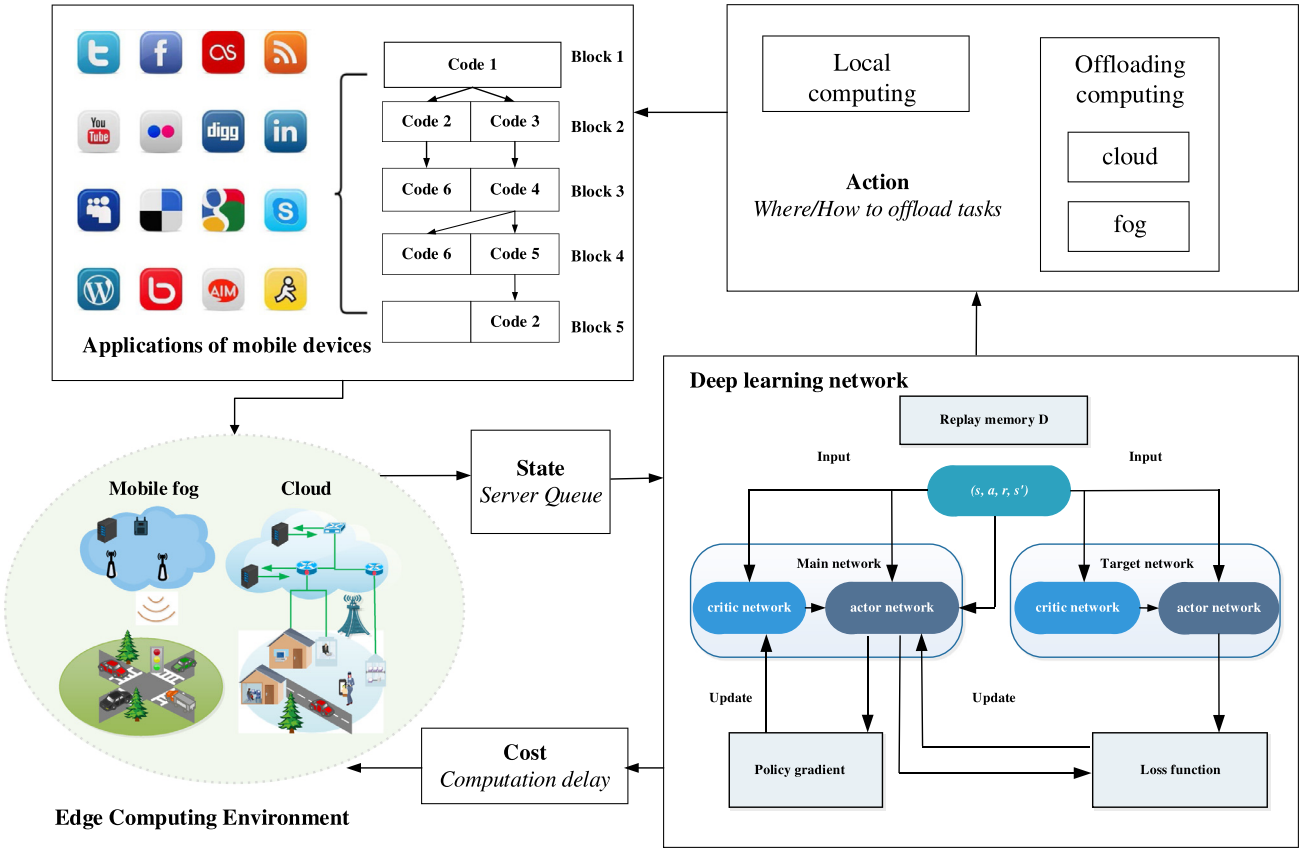

Fig. 5. The system model architecture.

Table 3
VM simulation parameters of mobile fog.

Nodes	Parameters	Values
Cloud node	Memory	64 GB
	CPU	2.0 GHz
	BW	8 Gbps
	Average hops from mobile station	13
	Total number	1
Mobile stations	Memory	2 GB
	CPU	1 GHz
	BW	200 Mbps
	Total number	30
Fog nodes	Memory	15 GB
	CPU	1.5 GHz
	BW	1 Gbps
	Average hops from mobile station	2
	Total number	8

order to ensure output action in $[0, 1]$ in the actor network, we employ the sigmoid function as the activation function. Meanwhile, the critic network, which has six fully connected layers and more units. Furthermore, add one tanh activation layer and two Relu activation layers in the critic network. More activations and layers can helpful to approximate the nonlinear Q-table. The simulation parameters of mobile fog are shown in Table 3.

For the cloud nodes network, the training offloading code blocks are 100, the capacity of replay buffer is 20 000, the mimi-batch is 60. In the critic network and the actor network, the learning rate is both set to 0.0002. The discount factor of Q-table is $\gamma = 0.9$, the update factor $\tau = 0.01$. Furthermore, the behavior noise value is 10. And the noise decay factor $\kappa = 0.9995$. The scale factors are set to $\lambda_1 = 100$ and $\lambda_2 = 5$, respectively.

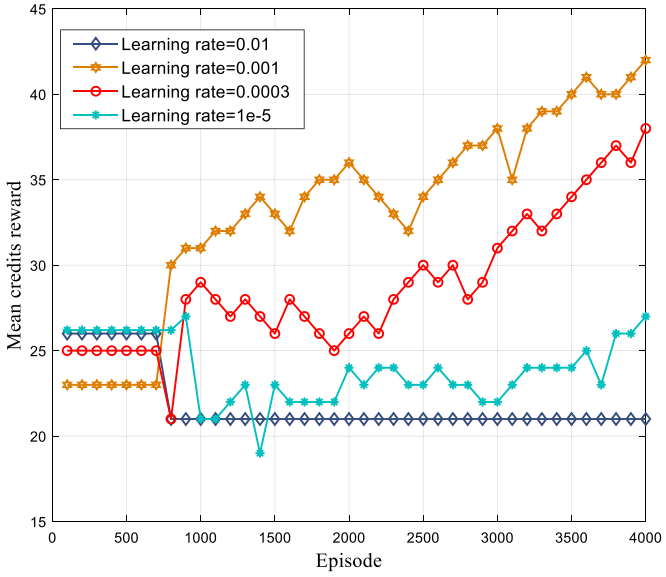

Fig. 6. Convergence performance of cloud with different learning rate.

For the mobile fog node, the training offloading code blocks are 200. In the critic network and the actor network, the learning rate are both 0.0005. The discount factor $\gamma = 0.9$.

The influence of parameters, for instance, the learning rate of the DDPG algorithm is shown in Figs. 6 and 7. From Fig. 6 we can see that, there is no good result when the learning rate is set to 0.01, because the policy became a greedy algorithm. When the learning rate is set to 0.001, the result of credit rewards fluctuates with the

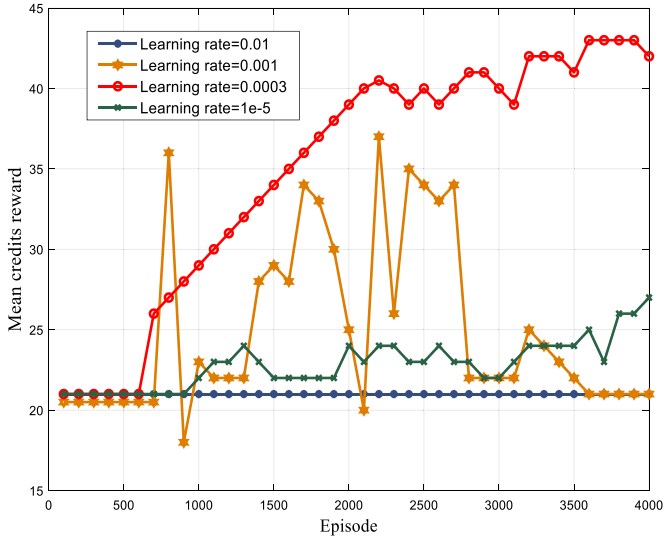

Fig. 7. Convergence performance of mobile fog network with different learning rate.

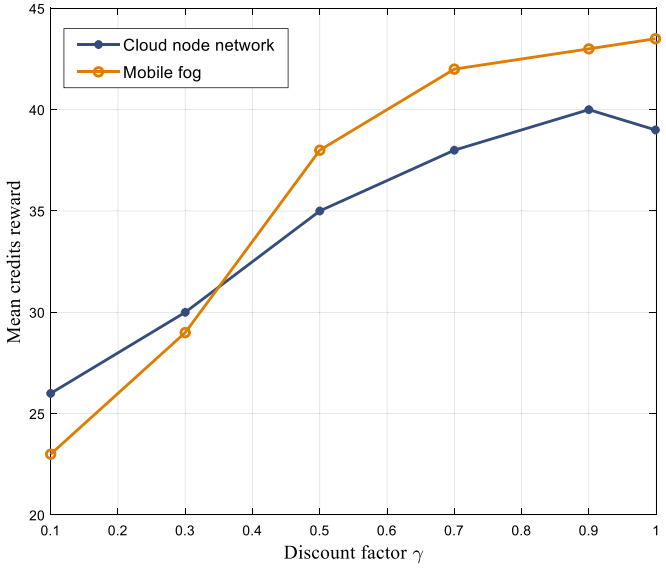

Fig. 8. The reward influence of mobile fog and cloud with different discount factor γ.

episode increment. Furthermore, the learning rate is set to 0.0003, the credits reward generally keeps improving with the episodes increment. However, when the learning rate is 10^{-5}, the reward grows slowly.

Unfortunately, cloud computing has an obvious disadvantage, that is, the communication distance from the mobile device to the remote cloud server is too long, resulting in high communication delay. Therefore, the mobile cloud is not enough to meet various wireless network applications with delay constraints and high computing power. Therefore, we will verify the performance of fog computing in Fig. 7.

In Fig. 7, similar to Fig. 6, when the learning rate is set to a large value, the credits reward will quickly obtain a bad result. Within a reasonable limit, the higher the learning rate can lead to faster growth of the results, but the greater the range of change. However, when the learning rate is smaller, the growth rate of performance decreases, the deviation will decrease. Hence, select the proper learning rate is more important, neither too small nor too large.

Fig. 8 shows the influence of the discount factor γ. For both cloud network and mobile fog network, when the discount factor is $\gamma = 0.9$ and $\gamma = 0.99$, the policy has the best performance. However, when γ too small or too large, the performance curve degrades. The reason is as follows. According to the bellman equation, the long-term average of exact expression is $\gamma = 1$. Hence, when γ is too small, the policy will focus more on immediate returns than on long-term rewards. Therefore, the appropriate γ value will improve the offload performance of mobile fog in the DDPG algorithm.

In Fig. 9, we consider four scenarios: (1) mobile fog, deep reinforcement learning algorithm with behavior noise; (2) mobile fog, deep reinforcement learning algorithm without behavior noise; (3) cloud, deep reinforcement learning algorithm with behavior noise; 4) cloud, deep reinforcement learning algorithm without behavior noise. As shown in Fig. 9, when training without the behavior noise, in both cloud network and mobile fog network, the mean credit rewards performance degrades. Compared with the behavior noise methods, the normalization of the state is much more important. The training is ineffective when without λ_1, λ_2, or without state normalization. However, it is difficult that learning a better result with a large value, because the randomly initialized NN tend to output a large value.

4.2. Performance comparison

We first compare the performance of offload computation reward among the proposed DDPG, DPG, Actor–Critic, and the PG algorithms under different nodes group. The results as shown in Figs. 10 and 11.

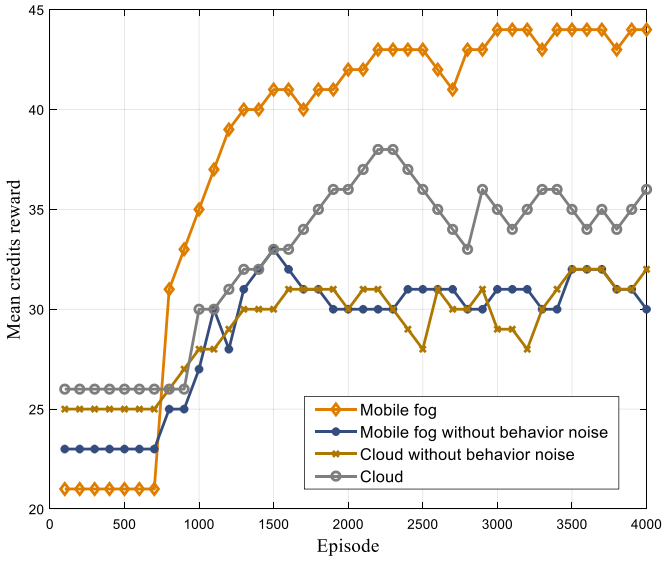

Fig. 9. Convergence performance of training without behavior noise.

For the mobile stations, we set the memory is 2 GB, the processor frequency is 1 GHz, bandwidth is 200 Mbps, the total number of mobile stations is 30. For the mobile fog, we set the memory is 15 GB, the processor frequency is 1.5 GHz, bandwidth is 1 Gbps, the total number of fog nodes is 8. Meanwhile, we use a cloud node, its memory is 64 GB, the processor frequency is 2.0 GHz, bandwidth is 8 Gbps. We measured the performance by following formulate:

$$\Phi(n) = [r_n - \tilde{r}(n)]/average(\tilde{r}(n)) \tag{31}$$

According to Figs. 10 and 11, we can see that our proposed scheme achieves at least 20%, 37%, 46% improvement on related performance compared with the policy gradient (PG), deterministic policy gradient (DPG) and actor–critic (AC) methods.

Next steps, we studied the response time and the system cost of three different computing environments, that is, mobile fog, cloud and smart devices. The resource usage can be modeled in the N-queen problem [45], where $N = \{5, 6, 7, 8\}$. When n > 8, the program execution time will be very long, so it is not suitable for mobile devices.

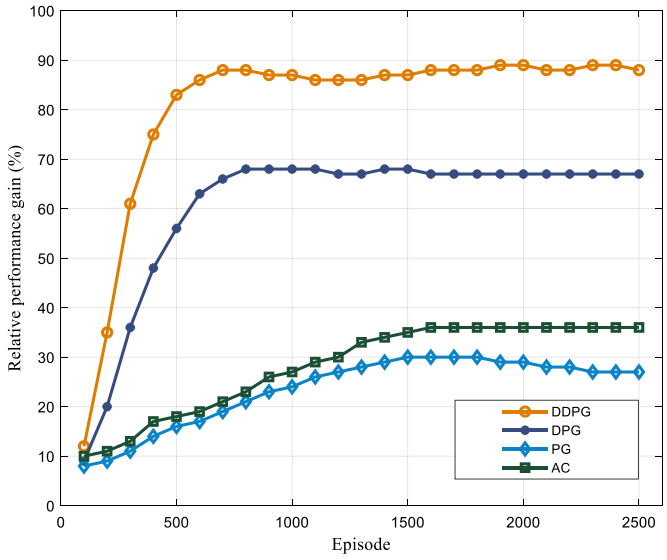

Fig. 10. Compare the relative performance of offload computation in mobile fog.

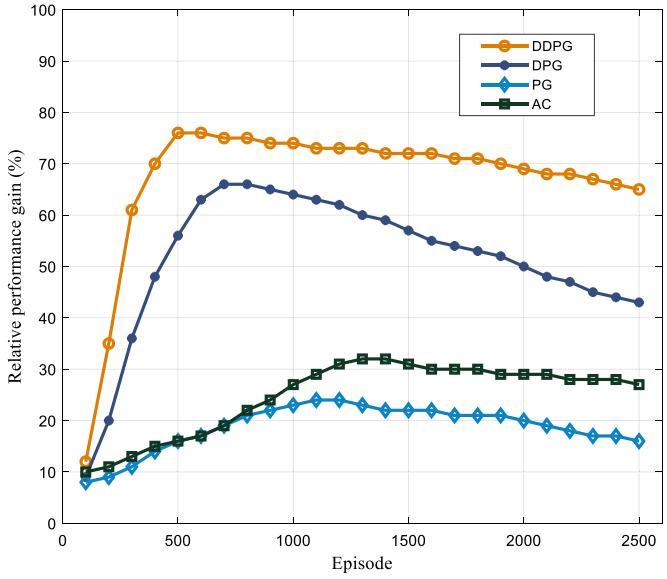

Fig. 11. Compare the relative performance of offload computation in cloud.

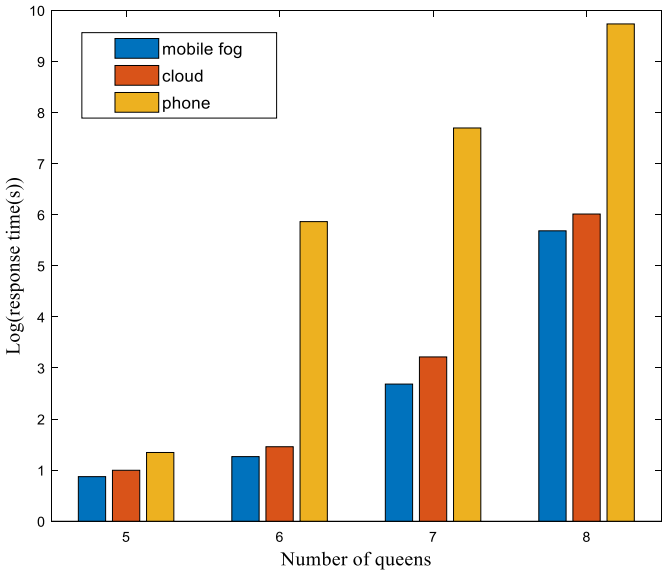

Fig. 12. The response times of mobile fog, cloud, and phone for benchmark N-queen problem.

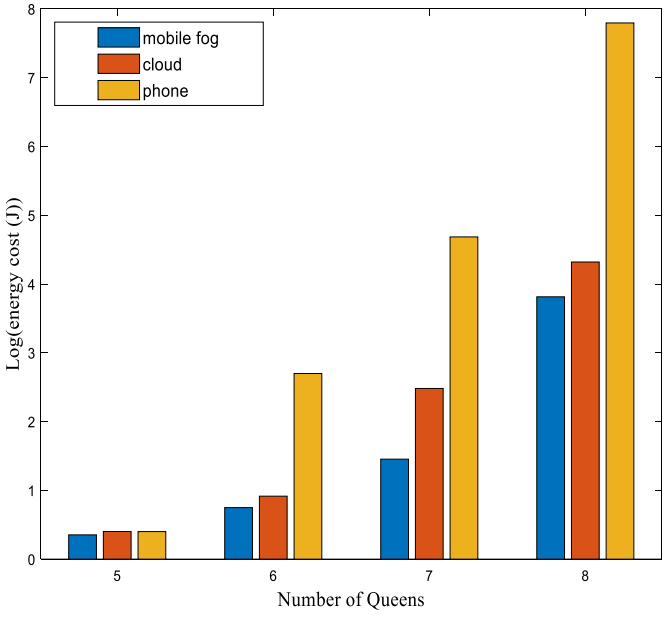

Fig. 13. For different number of Queens, energy cost of mobile fog, cloud and phone environment to generate outputs.

When n < 5, it is not suitable to calculate offloading because of the low calculation.

As shown in Fig. 12, the mobile fog costs less time to perform resource usage, however, the phones take a longer time to deal with the application. Compared with the smart device, for the performance of cloud servers, the cloud processes the solution also cost less time.

In Fig. 13, we can see that the results of energy cost in mobile fog, cloud and devices environment. The processor, peripherals and display unit of devices cost tremendous energy without offloading data. Whereas, the mobile fog and the cloud can work with less power consumption unit and without display. Due to the closest distance between the mobile fog nodes, the energy cost of fog is the lowest, which reduces the energy cost of wireless transmitting. For the remote cloud, the response time is higher than the mobile fog due to the remote cloud is always far from the devices and the mobile fog is closer to the BS.

The trend of cost is upward with locations number is increasing. However, more locations means that more computation requested and more computing resources are required to execute the tasks. Furthermore, we will evaluate the performance of fog resource provisioning. We use our learning-based scheme (DDPGF) to compare with the DRLF learning approach [46] and the traditional fog provisioning problem (FPP) approach [47]. Firstly, we consider the performance of fog resource provisioning allocation with different location numbers. As shown in Fig. 14(a), (b) and (c), we executed the experiments under three completion deadline situations: $D = \{130\ \text{ms}, 140\ \text{ms}, 150\ \text{ms}\}$.

From Fig. 14, we can see that all of the costs decrease when the task deadline increases. That is because when D increases, fewer computation resources are required and cost decreases.

Then, we compare the performances under different task arrival rates (TARs). From Fig. 15, we can see the trend of the system cost when TARs ranging from 5 to 12 task/s. The cost increases with the TARs increases, because more resources are needed to serve the increasing tasks. Through the experimental comparison between Figs. 13 and

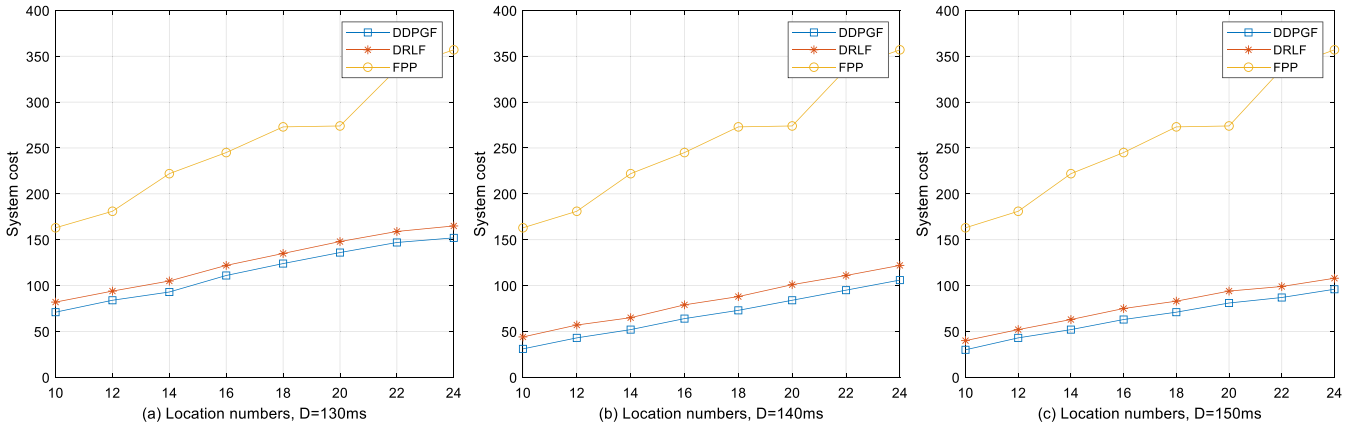

Fig. 14. The performance of fog resource provisioning under different locations number.

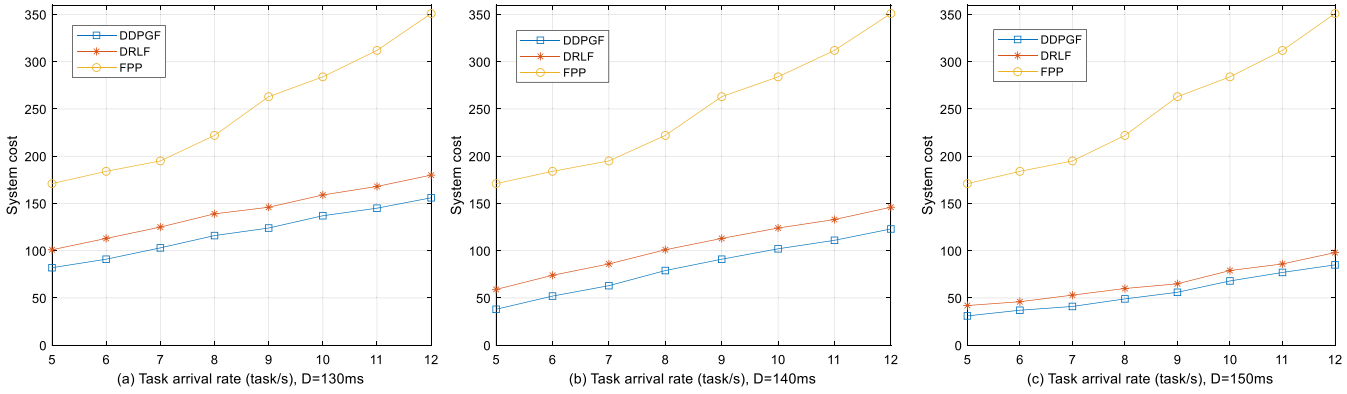

Fig. 15. The performance of fog resource provisioning under different task arrival rates.

15, we can see that the proposed algorithm can obtain lower system cost.

5. Conclusion and future work

The traditional mobile edge computing model is difficult to make the optimal offloading decision in complex networks, because finding the optimal strategy will inevitably encounter the problem of dynamic and changeable distribution state of the complex network. In this paper, we propose a learning-based mobile fog scheme to make full use of idle edge computing and storage resources and give an optimal solution to offload code blocks in the Internet of things. We modeled the offloading state sets as a Markov decision process (MDP) and used the DDPG algorithm to solve the problem of state space explosion and find an optimal policy to offload code blocks in different environments. Meanwhile, we elaborated the training procedure of the DDPG algorithm and designed a mobile fog model to realize offload computation. Furthermore, we performed the parametric analysis to find the influence of various algorithm parameters, such as discount factor γ and the learning rate. Under the different task arrival rates (TARs) and location numbers, the experimental results show the performance of fog resource provisioning of the proposed offloading scheme can improve offloading computation performance compared with PG, DPG and AC methods. In future work, we will focus on the privacy, virtualization, and mobility problem of mobile fog with learning-based schemes.

CRediT authorship contribution statement

Miaojiang Chen: Data curation, Software, Visualization, Writing - original draft of the current paper. **Tian Wang:** Formal analysis, Formal analysis of current paper. **Shaobo Zhang:** Supervision, Writing - review & editing. **Anfeng Liu:** Conceptualization, Methodology, Funding acquisition, Validation, Project administration, Resources of current paper.

Declaration of competing interest

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

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (No. 61772554, 62072475).

References

[1] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, et al., All one needs to know about fog computing and related edge computing paradigms: A complete survey, J. Syst. Archit. 98 (2019) 289–330.

[2] M. Yu, A. Liu, N. Xiong, T. Wang, An intelligent game based offloading scheme for maximizing benefits of IoT-edge-cloud ecosystems, IEEE Internet Things J. (2020) http://dx.doi.org/10.1109/JIOT.2020.3039828.

[3] X. Liu, H. Song, A. Liu, Unmanned aerial vehicle trajectory optimization for improved data collection in social networks, IEEE Trans. Netw. Sci. Eng. (2020) http://dx.doi.org/10.1109/TNSE.2020.3017556.

[4] S. Huang, A. Liu, S. Zhang, T. Wang, N. Xiong, BD-VTE: A novel baseline data based verifiable trust evaluation scheme for smart network systems, IEEE Trans. Netw. Sci. Eng. (2020) http://dx.doi.org/10.1109/TNSE.2020.3014455.

[5] X. Liu, P. Lin, T. Liu, T. Wang, A. Liu, W. Xu, Objective-variable tour planning for mobile data collection in partitioned sensor networks, IEEE Trans. Mob. Comput. (2020) http://dx.doi.org/10.1109/TMC.2020.3003004.

[6] Y. Liu, T. Wang, S. Zhang, X. Liu, X. Liu, Artificial intelligence aware and security-enhanced trace-back technique in mobile edge computing, Comput. Commun. 161 (2020) 375–386.
[7] X. Zhu, Y. Luo, A. Liu, M.Z.A. Bhuiyan, S. Zhang, Multi-agent deep reinforcement learning for vehicular computation offloading in IoT, IEEE Internet Things J. (2020) http://dx.doi.org/10.1109/JIOT.2020.3040768.
[8] S. Huang, Z. Zeng, K. Ota, M. Dong, T. Wang, N. Xiong, Intelligent Collaboration, An trust interconnections system for mobile information control in ubiquitous 5g networks, IEEE Trans. Netw. Sci. Eng. (2020) http://dx.doi.org/10.1109/TNSE.2020.3038454.
[9] A. Li, W. Liu, S. Zhang, M. Xie, Fast multicast with adjusting transmission power and active slots in software define IoT, IEEE Acceess (2020) http://dx.doi.org/10.1109/ACCESS.2020.3043762.
[10] J. Luo, X. Deng, H. Zhang, H. Qi, Qoe-driven computation offloading for edge computing, J. Syst. Archit. 97 (2019) 34–39.
[11] W. Huang, K. Ota, M. Dong, T. Wang, S. Zhang, J. Zhang, Result return aware offloading scheme in vehicular edge networks for 6g driving application, Comput. Commun. 164 (2020) 201–214.
[12] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, E. Riviere, Edge-centric computing: Vision and challenges, ACM SIGCOMM Comput. Commun. Rev. 45 (5) (2015) 2015.
[13] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, D.S. Nikolopoulos, Challenges and opportunities in edge computing, in: Proceedings of the IEEE International Conference on Smart Cloud, (2016) pp. 20–26.
[14] Y. Ouyang, A. Liu, N. Xiong, T. Wang, An effective early message ahead join adaptive data aggregation scheme for sustainable IoT, IEEE Trans. Netw. Sci. Eng. (2020) http://dx.doi.org/10.1109/TNSE.2020.3033938.
[15] M. Chen, T. Wang, K. Ota, M. Dong, et al., Intelligent resource allocation management for vehicles network: An A3c learning approach, Comput. Commun. 151 (2020) 485–494.
[16] T. Wang, H. Luo, X. Zeng, Z. Yu, A. Liu, A. Sangaiah, Mobility based trust evaluation for heterogeneous electric vehicles network in smart cities, IEEE Trans. Intell. Transp. Syst. (2020) http://dx.doi.org/10.1109/TITS.2020.2997377.
[17] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, K. Ha, The role of cloudlets in hostile environments, IEEE Pervas. Comput. 12 (4) (2013) 40–49.
[18] Z. Sanaei, S. Abolfazli, A. Gani, R. Buyya, Heterogeneity in mobile cloud computing: Taxonomy and open challenges, IEEE Commun. Surv. Tutor. 16 (1) (2014) 369–392.
[19] H. Dai, X. Zeng, Z. Yu, T. Wang, A scheduling algorithm for autonomous driving tasks on mobile edge computing servers, J. Syst. Archit. 94 (2019) 14–23.
[20] S. Wan, X. Li, Y. Xue, et al., Efficient computation offloading for internet of vehicles in edge computing-assisted 5g networks, J. Supercomput. 76 (4) (2020) 2518–2547.
[21] X. Zhu, Y. Luo, A. Liu, W. Tang, MZA. Bhuiyan, A deep learning-based mobile crowdsensing scheme by predicting vehicle mobility, IEEE Trans. Intell. Transp. Syst. (2020) http://dx.doi.org/10.1109/TITS.2020.3023446.
[22] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the Internet of Things, in: Proc. ACM MCC, Helsinki, Finland, (2012) pp. 13–16.
[23] S. Huang, J. Gui, T. Wang, X. Li, Mobile vehicles joint UAVs for secure data collection in smart city, Ann. Telecommun. (2020) http://dx.doi.org/10.1007/s12243-020-00798-9.
[24] N. Zhang, P. Dong, X. Wang, M. Obaidat, etc. al, When deep reinforcement learning meets 5g-enabled vehicular networks: A distributed offloading framework for traffic big data, IEEE Trans. Ind. Inf. 16 (2) (2020) 1352–1361.
[25] N. Zhang, P. Yang, J. Ren, D. Chen, Y. Li, X. Shen, Synergy of big data and 5g wireless networks: Opportunities, approaches, and challenges, IEEE Wirel. Commun. 25 (1) (2018) 12–18.
[26] T. Li, A. Liu, S. Zhang, T. Wang, N. Xiong, A trustworthiness-based vehicular recruitment scheme for information collections in distributed networked systems, Inf. Sci. 545 (2021) 65–81.
[27] N. Fernando, S.W. Loke, W. Rahayu, Mobile cloud computing: A survey, Future Gener. Comput. Syst. 29 (1) (2013) 84–106.
[28] M.H. Amini, H. Arasteh, P. Siano, Sustainable smart cities through the lens of complex interdependent infrastructures: Panorama and state-of-the-art, in: Sustainable Interdependent Networks, Vol. II, Springer, Cham, 2019, pp. 45–68.
[29] C.F. Liu, M. Bennis, H.V. Poor, Latency and Reliability-Aware Task Offloading and Resource Allocation for Mobile Edge Computing, in: Proc. 2017 IEEE Globecom Workshops (GC Wkshps), (2017) pp. 1-7.
[30] Z. Xiong, Y. Zhang, D. Niyato, et al., Deep reinforcement learning for mobile 5g and beyond: Fundamentals, applications, and chal-lenges, IEEE Veh. Technol. Mag. 14 (2) (2019) 44–52.
[31] S. Wan, L. Qi, X. Xu, et al., Deep learning models for real-time human activity recognition with smartphones, Mob. Netw. Appl. 25 (2) (2020) 743–755.
[32] A. Zhou, S. Wang, S. Wan, et al., LMM: latency-aware micro-service mashup in mobile edge computing environment, Neural Comput. Appl. (2020) http://dx.doi.org/10.1007/s00521-019-04693-w.
[33] S. Liu, G. Huang, J. Gui, T. Wang, X. Li, Energy-aware MAC protocol for data differentiated services in sensor-cloud computing, J. Cloud Comput. (2020) http://dx.doi.org/10.1186/s13677-020-00196-5.
[34] S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in: 2012 Proceedings IEEE INFOCOM, 2012, pp. 945–953.
[35] J. Benedetto, L. González, P. Sanabria, A. Neyem, J. Navón, Towards a practical framework for code offloading in the internet of things, Future Gener. Comput. Syst. 92 (2019) 424–437.
[36] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT press, Cambridge, 1998.
[37] V. Mnih, K. Kavukcuoglu, D. Silver, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[38] T.P. Lillicrap, J.J. Hunt, A. Pritzel, et al., Continuous control with deep reinforcement learning, in: Advances in International Conference on Learning Representations (ICLR), 2016.
[39] R. Sutton, D.A. Mcallester, S. Singh, Policy gradient methods for reinforcement learning with functionapproximation, in: Advances in Neural Information Processing Systems, 2000.
[40] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic Policy Gradient Algorithms, in: Proceedings of the 31 st International Conference on Machine Learning, (2014).
[41] Z.Z. Yu, M. Li, X. Xiao, J. Wang, Coordinated resource provisioning and maintenance scheduling in cloud data centers, in: INFOCOM, 2013 Proceedings IEEE, 2013, pp. 345–349.
[42] G. Weiz, Distributed reinforcement learning, Biol. Technol. Intell. Auton. Agents 144 (1995) 415–428.
[43] G. Weiz, Action selection and learning in multi-agent environments, in: Proceedings of the second international conference on from animals to animats 2: simulation of adaptive behavior: simulation of adaptive behavior, (1993) pp. 502-510.
[44] J. Sztrik, Basic.Queueing. Theory, Basic Queueing Theory, University of Debrecen: Faculty of Informatics, 2011.
[45] Y. Kwon, H. Yi, D. Kwon, S. Yang, et al., Precise execution offloading for applications with dynamic behavior in mobile cloud computing, Pervasive Mob. Comput. 27 (2016) 58–74.
[46] Y. Sun, M. Peng, S. Mao, Deep reinforcement learning-based mode selection and resource management for green fog radio access networks, IEEE Internet Things J. 6 (2) (2018) 1960–1971.
[47] O. Skarlat, S. Schulte, M. Borkowski, P. Leitner, Resource provisioning for IoT services in the fog, in: Proc. IEEE 9th Int. Conf. Service Oriented Comput. Appl. (SOCA), (2016) pp. 32–39.