Deep Reinforcement Scheduling for Mobile Crowdsensing in Fog Computing

Deep Reinforcement Scheduling for Mobile Crowdsensing in Fog Computing

HE LI, KAORU OTA, and MIANXIONG DONG, Muroran Institute of Technology, JAPAN

Mobile crowdsensing becomes a promising technology for the emerging Internet of Things (IoT) applications in smart environments. Fog computing is enabling a new breed of IoT services, which is also a new opportunity for mobile crowdsensing. Thus, in this paper, we introduce a framework enabling mobile crowdsensing in fog environments with a hierarchical scheduling strategy. We first introduce the crowdsensing framework that has a hierarchical structure to organize different resources. Since different positions and performance of fog nodes influent the quality of service (QoS) of IoT applications, we formulate a scheduling problem in the hierarchical fog structure and solve it by using a deep reinforcement learning based strategy. From extensive simulation results, our solution outperforms other scheduling solutions for mobile crowdsensing in the given fog computing environment.

Additional Key Words and Phrases: Fog Computing, Mobile Crowdsensing, Deep Reinforcement Learning

ACM Reference Format:
HE LI, KAORU OTA, and MIANXIONG DONG. 2018. Deep Reinforcement Scheduling for Mobile Crowdsensing in Fog Computing. *ACM Trans. Internet Technol.* 1, 1, Article 1 (January 2018), 20 pages. https://doi.org/10.1145/3141249

1 INTRODUCTION

Mobile crowdsensing is an emerging technology to collect data and information from individual mobile devices for different interests [14, 25, 31]. Mobile crowdsensing provides opportunities in many areas, such as public security, smart cities, disaster recovery and so on. For example, it is possible to precisely map noise pollution by using data from GPS receivers and microphones of smartphones [17].

A potential of mobile crowdsensing is collecting and analyzing multimedia data, e.g., images, videos, and voices, from smart devices to support graphically rich applications [23, 35, 46]. However, the nonnegotiable overhead in transferring and processing multimedia data limits applications of mobile crowdsensing, especially in real-time scenarios [21, 38, 49].

Fog computing offloads computing tasks from the cloud servers to fog nodes, which can relieve the pressure of processing multimedia data in mobile crowdsensing [8, 11]. Meanwhile, since fog nodes are usually much near to mobile devices, the network overhead in transferring

Authors' address: HE LI, heli@mmm.muroran-it.ac.jp; KAORU OTA, ota@mmm.muroran-it.ac.jp; MIANXIONG DONG, mxdong@mmm.muroran-it.ac.jp, Muroran Institute of Technology, Mizumoto Cho 27-1, Muroran, Hokkaido, 0508585, JAPAN.

1533-5399/2018/1-ART1 $15.00
https://doi.org/10.1145/3141249

large multimedia data is reduced [47]. Therefore, fog computing is a promising technology for mobile crowdsensing applications with multimedia information [5].

However, task scheduling is a perennial issue in most modern computing systems, and fog computing is not exempt [47]. The complex structure of fog computing increases the difficulty in scheduling mobile crowdsensing tasks [9]. Meanwhile, multimedia tasks are different from traditional tasks, which needs different hardware such as graphics processing units (GPUs) and computing models such as neural networks [26, 27]. Most existing scheduling techniques are not acclimatized for scheduling mobile crowdsensing tasks in fog environments [10].

Most mobile crowdsensing applications will use hierarchical procedures in processing multimedia information from unstructured data to textual information [2]. For example, in processing images, applications will first recognize the objects in images and then execute further computing [13]. In image recognition, an image data will be preprocessed into different features for classification [40]. Fog computing has a hierarchical structure which is appropriate for multimedia crowdsensing application [41]. For example, mobile devices execute the collecting and coding, fog nodes execute preprocessing, and cloud servers execute further computing [20]. Therefore, in this paper, we state a scheduling problem to optimize the quality-of-service (QoS) of both mobile users and crowdsensing providers in this hierarchical structure.

Furthermore, since there is no given structure or topology of fog computing, the scheduling method needs more scalability and elasticity for mobile and heterogeneous environments. Most scheduling methods focus on given structures such as fat-tree structures in data centers and mesh structures in ad-hoc networks [1, 12, 29, 50]. We introduce a deep reinforcement learning model into our hierarchical scheduling to self-adapt different structures and topologies of fog computing after extensive training. In the performance evaluation, we compare the performance of our solution and other methods in different fog computing structures, and the results show the reinforcement learning concept improves the scheduling efficiency in varying environments.

The main contributions of this paper are summarized as follows.

- We first state the scheduling issue of mobile crowdsensing tasks in fog computing with multimedia data. We investigate the hierarchical structure of both fog computing and multimedia processing.
- We then introduce deep reinforcement learning into the scheduling problem and propose a scalable and elastic method to self-adapt varying fog computing structure and mobile environments.
- We take the performance evaluation of our solution and other scheduling methods with different settings of fog computing. From extensive simulation results, the proposed deep reinforcement learning model shows good efficiency for scheduling crowdsensing tasks in fog computing.

The remainder of this paper is organized as follows. Section 2 discussed the related works of task scheduling in fog computing and reinforcement learning. Section 3 first presents the hierarchical structure of fog computing and mobile crowdsensing, and then state the scheduling problem. The deep reinforcement learning based scheduling solution is described in Section 4. Section 5 evaluates the proposed scheduling method through extensive simulations. Finally, this paper is concluded with future works in Section 6.

2 RELATED WORK

In this section, we first introduce some recent works focusing on scheduling problems in fog computing, and then discuss the technology of reinforcement based scheduling.

2.1 Scheduling Tasks in Fog Computing

Scheduling is an essential issue in fog computing because of complex and sophisticated resource management. As well as cloud computing, it is important to provide scheduling tools in fog computing [22]. Hong et al. proposed a programming model to support fog computing in the Internet of Things (IoT) environment [19]. As a basic function, the programming model provides high-level management interfaces for supporting programmable scheduling. Users can develop different scheduling to improve the efficiency of resource management with this programming model in fog computing.

Some fog computing systems are built by virtualization for good encapsulation and migration. In virtualization-based fog computing such as GigaSight [36] and ThinkAir [24], virtual machine placement and movement are usually the main approaches for scheduling resources. However, most lightweight fog computing systems are hard to afford the additional overhead brought by virtualization. Thus, our solution focuses on a compatible task scheduling methodology for different implementations of fog computing.

Unlike general cloud computing, mobility is a critical issue in fog computing where few general scheduling methods work correctly in a mobile environment. Bittencourt et al. proposed a mobility-aware task scheduling in fog computing, which focused on the scheduling consistency in mobile environments [8]. The scheduling method implemented an applicant in each access point to maintain the task information during user movement. Our work also includes the consideration of user mobility with a more efficient scheduling method.

Fog computing is usually deployed in heterogeneous environments which increases the difficulty of resource management and task scheduling. Bossche et al. first introduced the directed acyclic graph (DAG) into the task scheduling problem in a hybrid cloud environment which is similar to the fog computing [7]. Thus, Pham et al. introduced DAG to formulate the heterogeneous resources in fog computing and proposed a simple scheduling algorithm to improve the efficiency of the task scheduling in fog computing [34].

More mathematical formulations are introduced for fog computing with heterogeneous resources or user mobility. Sharma et al. introduced a probabilistic model into the convergence of IoT and fog computing to improve the resource utilization efficiency [37]. The probabilistic model is used to understand traffic features and dynamics in the heterogeneous IoT environment. Meanwhile, a cooperative-based model is proposed to schedule IoT tasks in fog computing and improve the quality of data collection [28]. The model focused on the cooperation of mobile participants in the suburban regions. Based on the cooperative relationship, it is possible to distribute smart sensors according to the population density. In our work, we propose an elastic reinforcement learning method for the compatibility of different heterogeneous and mobile environments instead of static mathematical formulations.

Since edge computing and fog computing have similar hierarchical structure, scheduling methods in edge computing also provide some experiences to our work. Li et al. provide a hierarchical scheduling for deep learning based IoT applications in edge computing, which divided deep learning networks into different offloading layers [**?**]. Thus, it is possible to offload a part of each deep learning task instead of the entire task into the edge node.

2.2 Reinforcement Learning based Scheduling

Reinforcement learning is a promising scheduling method in many real-time systems [18, 44, 45]. Usually, reinforcement learning is an efficient solution for a Markov decision process (MDP) problem. In a general dynamic system, the outcome of each decision is partly random and partly under control. MDPs model decision making in the dynamic system and the decision-making problem can be transferred to an MDP problem, and a wide range of optimization problems include task scheduling can be formulated as MDP problem.

Since it is very difficult to solve complex MDP problems, some early works try to apply reinforcement learning in some simple scheduling problems such as multi-agent scheduling and job-shop scheduling [3, 48]. These problems usually have two layers for task scheduling and reinforcement learning, which is less than the number of layers in some ordinary fog computing environments.

Glaubius et al. proposed a reinforcement learning based real-time scheduling in cyber-physical systems [18]. The scheduling model can improve the system performance even with limited information of tasks. The task scheduling model is formulated as an MDP problem and solved via a successive elimination algorithm proposed by Even-Dar et al [16]. However, this approach only considered the relationship between tasks and jobs in resource management while there are more problems such as mobility, localization, and heterogeneity in fog computing.

Deep reinforcement learning brings new opportunities to the scheduling problems in complex environments. Deep reinforcement learning achieved dramatically better results than ordinary reinforcement approaches in some complex problems such as Atari games and Go [32, 39]. Since deep reinforcement learning introduces deep neural networks to learn representations from high-dimensional inputs, it is possible to solve problems in some scenarios where making decisions will affect the state of the entire environment.

DeepRM is the first solution applying deep reinforcement learning for resource management [30]. The authors transfer the resource and task management states into images as the input of the deep neural network, which is more feasible for complex resource management problems. Since the deep neural network has a higher capability for learning states from complex resource management problems, some works introduced deep reinforcement learning for scheduling resources in large-scale network systems [33, 42].

Atallah et al. proposed a deep reinforcement learning-based model in vehicular networks to extend the lifetime of roadside units (RSUs) as well as to guarantee the quality of service (Q levels [4]. The authors applied a traditional full-connected artificial neural network in the deep reinforcement learning model. The stochastic gradient descent (SGD) method is adopted for training the neural network.

However, the proposed model is not appropriate for a realistic environment due to the hidden-layer based neural network design. In our work, we introduce a four layer convolutional neural network (CNN) based model for the fog computing environment.

3 SCHEDULING PROBLEM

In this section, we first describe the hierarchical structures of fog computing and mobile crowdsensing tasks. Then, we apply the MDP to formulate the mobile crowdsensing task scheduling problem in fog computing.

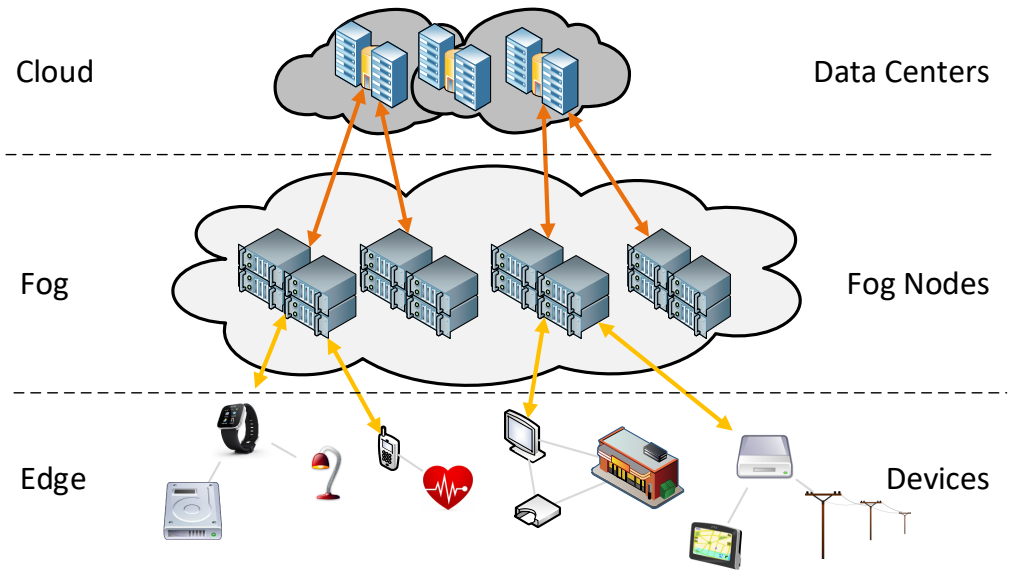


Fig. 1. Hierarchical Structure of Fog Computing

3.1 Hierarchical Structure

Usually, fog computing has a hierarchical structure as shown in Fig. 1 including a cloud layer, a fog layer, and an edge layer. In the cloud layer, cloud servers in data centers execute the major computing of each service. In mobile crowdsensing, data analytics are usually deployed in the cloud layer since sensed datasets are stored in the cloud servers. In the edge layer, there are many types of smart devices such as smartwatches, smartphones, personal computers and smart meters. These devices will sense different data in a mobile crowdsensing scenario.

In traditional computing structures, all collected data will be directly sent to the cloud layer for further processing. There are two problems in traditional cloud computing in the mobile crowdsensing scenario. One problem is that the performance of network connections between the cloud layer and the edge layer limits the amount of sensed data and increases the latency in real-time analytics. Another problem is that the cloud servers are hard to process all sensed data including some multimedia data.

Fog computing adds a fog layer between the cloud layer and the edge layer to offload computing from the cloud layer as well as to preprocess the primary sensed data from the edge layer. Fog nodes in the fog layer are usually deployed in the transmission paths between edge devices and cloud servers and are closer to the edge layer. For example, carriers can deploy the fog nodes in radio access networks (RANs) to reduce the traffic from RANs to core networks.

In mobile crowdsensing, fog computing can improve the efficiency of data analytics and reduce the inbound traffic of the cloud layer. As shown in Fig. 2, we use an example to introduce the profits of fog computing for mobile crowdsensing. Flower recognition is a

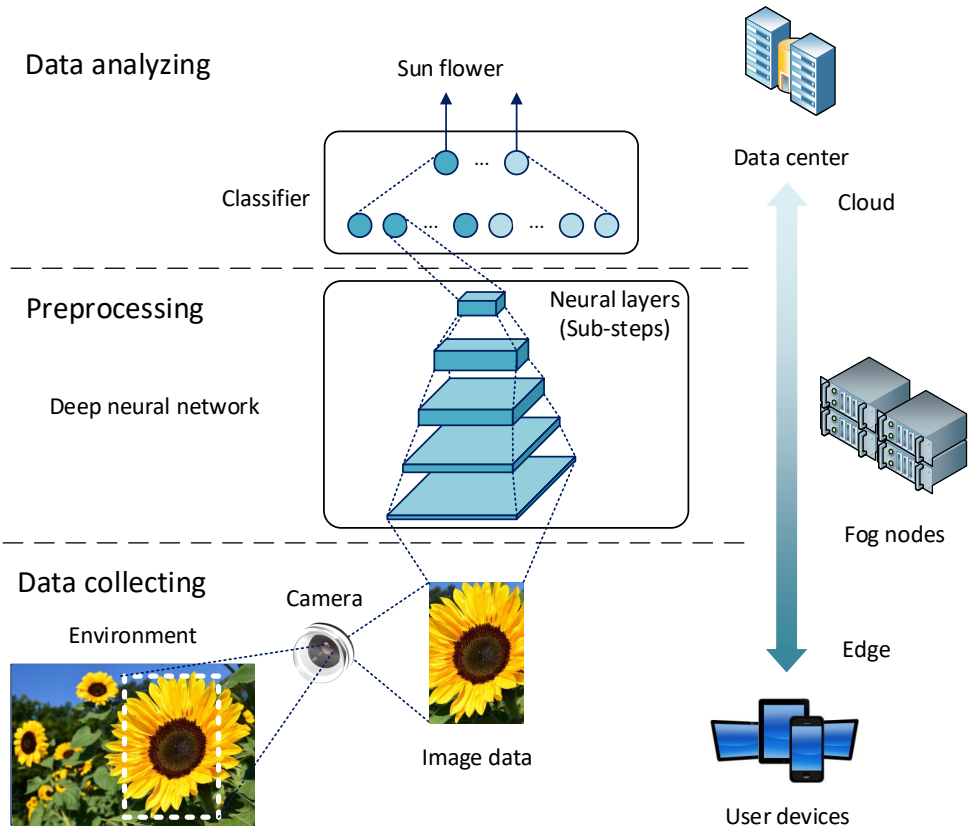


Fig. 2. Hierarchical structure of a mobile crowdsensing task

popular application that some people want to know more about the environment. The flower recognition application is very precise in recognizing different flowers and also provide a good opportunity for mobile crowdsensing. When users upload flower photos to the cloud layer, it is possible to analyze more information such as positions, flowering seasons and vegetations.

However, analyzing image data is a costly task for the cloud servers. The size of image data is also very large for transmission from devices to the cloud layer. Thus, in fog computing, fog nodes first preprocess image data and extract some textual information and then transfer the extracted information to the cloud layer, which improves the network efficiency and relieve the computation pressure of cloud servers.

Most image recognition applications adopt deep learning networks to extract image features for classification. The deep learning-based tasks also have a hierarchical structure in processing procedures including a data collecting step, a preprocessing step and a data analyzing step. For the flower recognition application, edge devices record and code the flower image data in the data collecting step. In the preprocessing step, deep neural network-based learning modules extract features from the image data. In the data analyzing step,

classifiers recognize the flower from the extracted features. Moreover, in the preprocessing step, learning modules have multiple layers which can be divided into small sub-steps for extracting features.

Therefore, whole or part of the preprocess step of a given mobile crowdsensing task can be executed in any layers of fog computing. For example, edge devices can preprocess the collected data before transferring with higher energy consumption. As a result, the mobile crowdsensing task scheduling is a two-dimensional decision problem to decide the position and layer for executing each step and sub-step in fog computing.

3.2 Problem Statement

In the task scheduling model, we use set U to denote all users and u_i, $i \in \{1, |U|\}$ to denote a user in U. Each user u_i has a sequence of tasks $t_{i,j}|_{j=1}^{j=\infty}$. We first define indicator $X_{i,j}$ to denote whether a task is fulfilled, given by

$$X_{i,j} = \begin{cases} 1, & \text{if } l_{i,j} \leq L_i, \\ 0, & \text{elsewise}, \end{cases} \tag{1}$$

where $l_{i,j}$ is the time to completion (TTC) of task $t_{i,j}$, and L_i is the required TTC of user u_i.

If the fog computing system can complete a task before required TTC, the task is fulfilled. There are many tasks of each user and it is hard to fulfill all tasks in the scheduling with limited resources. Thus, for user u_i, we use the service fulfillment ratio Q_i to denote QoS, defined as

$$Q_i = \frac{\sum_{j=1}^{N_i} X_{i,j}}{N_i} \tag{2}$$

where N_i is the number of completed tasks of user u_i.

Task $t_{i,j}$ has a sequence of steps $s_{i,j,k}|_{k=1}^{k=K_{i,j}}$ where $K_{i,j}$ is the number of steps in task $t_{i,j}$. In sequence $s_{i,j}$, step $s_{i,j,k+1}$ will start after receiving all output data of step $s_{i,j,k}$. For step $s_{i,j,k}$, there is an input data size $d^I_{i,j,k}$, an output data size $d^O_{i,j,k}$, and TTC $c_{i,j,k}$. We assume there is no branch in tasks so that $d^O_{i,j,k} = d^I_{i,j,k+1}$.

For a given fog computing system, we use set F to denote all fog nodes and f_m to denote a node in set F. The TTC of task $t_{i,j}$ is calculated as

$$l_{i,j} = \sum_{k=1}^{K_{i,j}-1} \left(\frac{d^O_{i,j,k}}{B_{i,j,k+1}} + c_{i,j,k} + \delta_{i,j,k+1}\right) + c_{i,j,K_{i,j}} + w_{i,j} + v^I_{i,j} + v^O_{i,j} \tag{3}$$

where $B_{i,j,k+1}$ is the bandwidth for step $s_{i,j,k}$, $\delta_{i,j,k+1}$ is the latency from step $s_{i,j,k}$ to step $s_{i,j,k+1}$, $w_{i,j}$ is the waiting time for scheduling, $v^I_{i,j}$ is the time for inputing data to step $s_{i,j,1}$, and $v^O_{i,j}$ is the time for inputing preprocessed data to the cloud.

In task scheduling, the scheduler usually slices time into time slots. We use τ to denote the length of each time slot. We use $b_{i,j}$ to denote the begin time of task $t_{i,j}$. In time slot n, we use a set F^a_n to denote the available fog nodes, and a set T^r_n to denote the tasks for scheduling, given by

$$T^r_n = \bigcup_{b_{i,j}=n} \{t_{i,j}\}. \tag{4}$$

We analyze the impact of the task scheduling with rewards and costs in time slot n. Let A_n denote the scheduling decision set and $a_{i,j,n}$ denote the decision for task $t_{i,j}$ where $b_{i,j} = n\tau$. Decision $a_{i,j,n} = \{f^a_{i,j,1}, f^a_{i,j,2}, ..., f^a_{i,j,K_{i,j}}\}$ denotes assigned fog nodes for each step

of task $t_{i,j}$ in time slot n. In time period during the execution of step $s_{i,j,k}$, the assigned fog node is not available, given by

$$f^a_{i,j,k} \notin F^a_{n'} \tag{5}$$

where $n' = [n + \delta_{i,j,k-1} + \frac{d^I_{i,j,k}}{B_{i,j,k}}, n + \delta_{i,j,k-1} + \frac{d^I_{i,j,k}}{B_{i,j,k}} + c_{i,j,k}]$, and $\delta_{i,j,0} = 0$.

In the task scheduling, since the bandwidth and latency between fog nodes and users are varying because of the user mobility, time $v^I_{i,j}$ is related to user u_i and time slot n, given by

$$v^I_{i,j} = V(f^a_{i,j,1}, n) \tag{6}$$

where $V(\cdot)$ is a function to calculate the total time for inputting data to step $s_{i,j,1}$ from user u_i in time slot n.

The bandwidth and latency for step $s_{i,j,k}$ are decided by fog node $f^a_{i,j,k-1}$ and $f^a_{i,j,k}$, given by

$$B_{i,j,k} = B^f(f^a_{i,j,k-1}, f^a_{i,j,k}), \text{ and} \tag{7}$$

$$\delta_{i,j,k} = \delta^f(f^a_{i,j,k-1}, f^a_{i,j,k}) \tag{8}$$

where function $B^f(\cdot)$ and $\delta^f(\cdot)$ denote the bandwidth and latency between two servers, respectively.

We use X_n to denote the scheduling state including assigned and available fog nodes from time n to $n + max(n^a)$, where n^a is the maximum TTC of steps assigned before time n.

The cost paid at time n consists of three components, the TTC of each task in set T^r_n, the expected bandwidth cost for submitting the output data after the executing tasks in set T^r_n in fog nodes, and the amount of time for processing tasks in the cloud layer. The reward of the mobile crowdsensing task is the uploaded information. In task scheduling, the reward of task $t_{i,j}$ is equal to $d^O_{i,j,K_{i,j}}$. Meanwhile, time $v^O_{i,j}$ is decided by fog node $f^a_{i,j,K_{i,j}}$, given by

$$v^O_{i,j} = \frac{d^O_{i,j,K_{i,j}}}{B^f(f^a_{i,j,K_{i,j}}, \mathbf{C})} \tag{9}$$

where $\mathbf{C}$ is the cloud layer.

Thus, we use reinforcement learning to maximize the total reward of the fog computing system. Let r_n denote the single step reward of the fog computing system, and let R_n denote the total future discounted reward, given by

$$R_n = \sum_{t=n}^{N} \gamma^{t-n} \cdot r_n \tag{10}$$

where N is a target execution time of the fog computing system, γ is the discount factor. Discounted factor γ is usually less than 1 in typical MDP formulation to make discounted reward R_n converge.

There are three goals in the task scheduling as follows.

- The scheduling guarantees the QoS of each user.
- The bandwidth cost from fog nodes to the cloud layer is minimized.
- The cloud layer executes minimum computing.

In mobile crowdsensing, each piece of user data, upload traffic and cloud resources can be priced reasonably. Thus, we design a reward r_n for all costs of bandwidth and computing at time n as

$$r_n = \sum_{i=1}^{|U|}(\alpha \cdot d^O_{i,j,K} \cdot X_{i,j} - \beta \cdot \sum_{k=2}^{K_{i,j}} B_{i,j,k} - \eta \cdot \sum_{k=1}^{K_{i,j}} c_{i,j,k}) \tag{11}$$

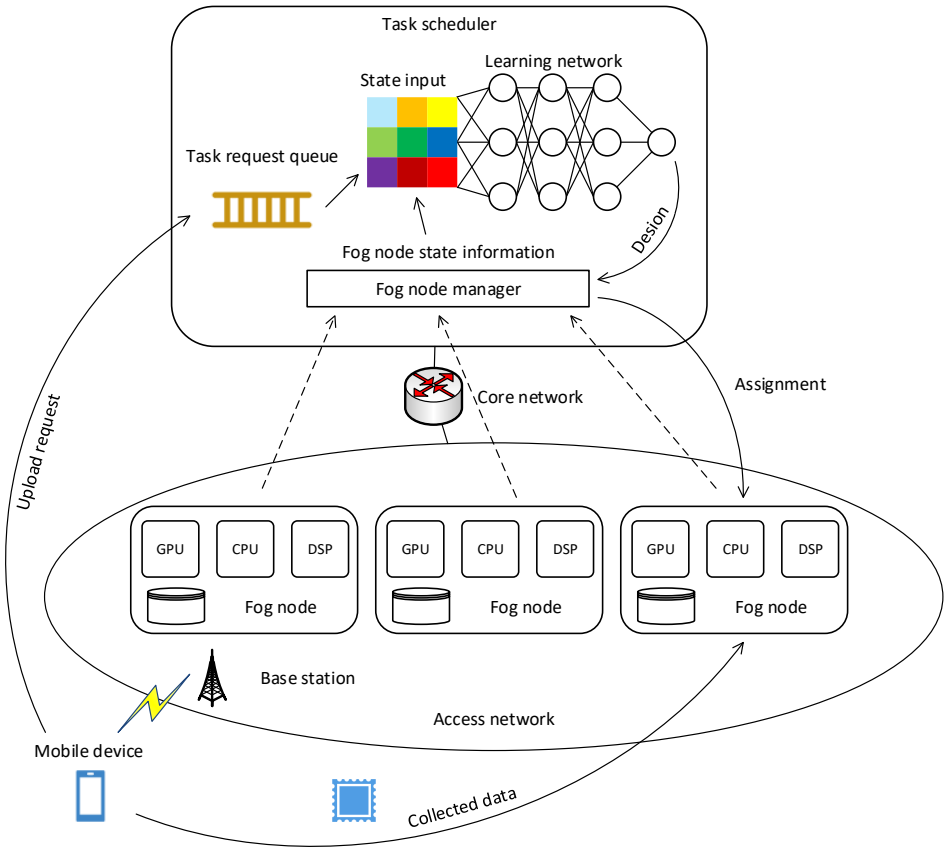


Fig. 3. Processes of the deep reinforcement scheduling

where α, β and η are the unit price of user data, upload traffic, and fog processing, respectively.

Task scheduling problem for mobile crowdsensing in fog computing: given a set of fog nodes and a set of users, the task scheduling problem attempts to assign fog nodes to steps of each task submitted by users. In the scheduling, QoS of each user is guaranteed with minimal computing and bandwidth costs.

4 DEEP REINFORCEMENT SCHEDULING

For solving the task scheduling problem in fog computing, we propose a deep reinforcement scheduling solution shown in Fig. 3. We add a task scheduler to the cloud layer to decide the scheduling strategy for the fog computing. The task scheduler consists of a task request queue, a learning-based scheduling decider, and a fog node manager. When a mobile user wants to upload data for mobile crowdsensing, the mobile device sends a request to the task scheduler. The task scheduler will put all requests into a request queue for further scheduling. If waiting time $w_{i,j}$ of task $t_{i,j}$ in the request queue exceeds $L_i - v'_{i,j}$ where $v'_{i,j}$

is time for inputing data from u_i to the cloud layer, we assign the cloud layer for processing task i, j to meet the QoS requirement.

In task scheduling, the scheduler first collects the state information of all fog nodes and task requests and then generates several scheduling decisions. For each decision, the scheduler makes a state bitmap as the input of the learning network. The learning network inferences the input and generates values of input scheduling decisions. The task scheduler selects and sends the decision with the best value to the fog node manager. The fog node manager assigns the fog nodes in the decision of the task request. After scheduling, the mobile device uploads data to the fog node assigned for the first step of the required task.

The optimal scheduling needs a determined stationary policy denoted by π that which action A_n should be applied at time n for maximizing the reward and minimizing the cost. Decision set π is denoted by a sequence of functions $\{A_1, A_2, A_3, ..., A_N\}$ where A_n is a function of the state X_n. We use set $\mathbb{A}_n$ and $\mathbb{X}_n$ to denote all admissible decisions and states at time n, and $\mathbb{A} = \{\mathbb{A}_1, \mathbb{A}_2, ..., \mathbb{A}_N\}$ and $\mathbb{X} = \{\mathbb{X}_1, \mathbb{X}_2, ..., \mathbb{A}_N\}$ to denote decisions and states in whole scheduling period. The goal of the scheduling is to find an optimal set π^* to maximum total discounted rewards. When the scheduling is following set π, the decision at time n is $A_n = \pi(X_n)$. Thus, the reward of this decision is $r_n = r(X_n, \pi(X_n))$ and the optimal decision set is given by

$$\pi^* := \arg\max_{\pi \in \Pi} R_n^{\pi}, \text{ and} \tag{12}$$

$$R_n^{\pi} = \sum_{t=n}^{N} \gamma^{t-n} \cdot r(X_n, \pi(X_n)) \tag{13}$$

From reinforcement learning model [43], we use a $Q^*(\cdot)$ function to define the decision-reward function defined by the maximum reward in state X_n with an optimal decision A_n. We get the $Q^*(X_n, A_n)$ from the Bellman optimality equation [6] as

$$Q^*(X_n, A_n) = \mathbb{E}_{X_{n+1}}[r_n + \gamma \cdot \max_{A_{n+1}}[Q^*(X_{n+1}, A_{n+1})]|X_n, A_n] \tag{14}$$

where $\mathbb{E}$ is the expectation.

Traditional reinforcement learning will estimate the Q^* function iteratively with the Bellman equation as

$$Q_{i+1}(X_n, A_n) = \mathbb{E}_{X_{n+1}}[r_n + \gamma \cdot \max_{A_{n+1}}[Q_i(X_{n+1}, A_{n+1})]|X_n, A_n] \tag{15}$$

where i is the number of iterations.

Thus, the optimal Q^* function will be covered by iterations of (15) as

$$Q_i \to Q^* \text{ as } i \leftarrow \infty. \tag{16}$$

However, the iterations are not practical since there is no general way to find the Q function which is estimated separately with different π. It needs function approximator to estimate Q^* function as

$$Q(X, A; \theta) \approx Q^*(X_n, A_n) \tag{17}$$

where θ is a sequence of approximation weights.

Traditional reinforcement learning usually adopts a linear function approximator while in deep reinforcement learning, the approximator is non-linear with a deep neural network. In our solution, we organize the approximation weights θ as a Q-network to find the function approximator. We define a loss function $\mathbb{L}_i(\theta_i)$ that changes at the ith iteration as

$$\mathbb{L}_i(\theta_i) = \mathbb{E}_{\rho(X_n, A_n)}[(Y_i - Q(X_n, A_n; \theta_i))^2] \tag{18}$$

where

$$Y_i = \mathbb{E}_{X_{n+1}}[r_n + \gamma \cdot Q(X_{n+1}, A_{n+1}; \theta_{i-1})|X_n, A_n] \tag{19}$$

is the target of the ith iteration and $\rho(X, A)$ is a behavior distribution over sequences X and decisions A. We can iteratively train the Q-network by minimizing a sequence of (18). We maintain fixed parameters from the $(i-1)$th iteration in optimizing (18) in the ith iteration. Since target Y_i depends on the value of approximation weight θ_{i-1}, we use a gradient to differentiate (18) as

$$\begin{aligned}\nabla_{\theta_i} \cdot \mathbb{L}_i(\theta_i) = \mathbb{E}_{\rho(X_n, A_n)}[(r_n + \gamma \cdot Q(X_{n+1}, A_{n+1}; \theta_{i-1}) \\ -Q(X_n, A_n; \theta_i)) \cdot \nabla_{\theta_i} \cdot Q(X_n, A_n; \theta_i)].\end{aligned} \tag{20}$$

We use the stochastic gradient descent (SGD) to optimize (18) instead of computing the full expectations in (20) for simplifying the computation procedure. For building an off-policy solution, we apply an ϵ-greedy strategy to explore $\rho(X, A)$. In the exploration, the method learns the policy as

$$A = \arg\max_A Q(X, A; \theta) \tag{21}$$

with probability $1 - \epsilon$ and selects a random decision with probability ϵ.

As shown in Algorithm 1, we introduce the algorithm of the deep reinforcement scheduling for mobile crowdsensing in fog computing. We first initial a replay buffer R to store transitions for mini-batch sampling. The buffer R is used for storing previous experience during learning the deep reinforcement learning model. Due to the low variance in immediate transitions, it is necessary to use past experience to make the reinforcement learning model converge quickly.

A Q-network with random θ is built for the learning procedures. The algorithm performs M iterations to find the optimal scheduling decisions. At the beginning of each iteration, the algorithm will generate a state without any tasks. Then, the algorithm schedules tasks from time 1 to N. In the scheduling, an ϵ-greedy strategy is performed to generate the decision A_n. Thus, the reward is calculated with (11), and all fog nodes are assigned with decision A_n. The algorithm calculates the new state X_{n+1} with assigned fog nodes from time $n+1$ to $n+1+max(n^a)$. The transition (X_n, A_n, r_n, X_{n+1}) is stored in buffer R. Then, the algorithm loads $|B|, 1 \leq |B| \leq k$ transitions from buffer R for the mini-batch sampling. In the sampling, if the episode terminates, the output is set as r_i otherwise $r_i + \gamma \cdot \max_{A_{i+1}} Q(X_{i+1}, A_{i+1}; \theta_{n-1})$. In sampling, the value of θ_n is updated by performing the SGD. After sampling, the value of θ_{n-1} is updated by θ_n.

In the practical application, we build a CNN to solve the scheduling problem. For input of the CNN network, we design a bitmap structure to describe the fog node scheduling. In a fog computing environment, each fog node is geographically distributed with different network architectures. Although the distance between users and fog nodes is varying due to user mobility, the connections between fog nodes are usually stable. Thus, we assume $\max(L_i) \leq |F|$ and design a matrix of size $[|F| \times |F|]$ as the input of the CNN network. In the fog node assignment, steps of the same task will be assigned to the same fog node or neighbor fog nodes. Thus, we use same value to color fog nodes executing steps from the same task in the matrix. As shown in Fig. 4, fog nodes f_1, f_2 and f_3 at time 1, 3, 5 are colored by red when executing step $s_{1,j,n}$, $s_{1,j,n+2}$ and $s_{1,j,n+5}$ from user u_1's task $t_{1,j}$.

With the input data structure, we build a four-layers neural network including two convolution layers and two fully connected layers. As shown in Fig. 5, we use an example to describe the structure of the deep learning network. In the example, we use a 64x64x4 bitmap image with the input structure, which means there are 64 fog nodes and $\max(L_i) = 64\tau$.

ALGORITHM 1: Deep Reinforcement Scheduling with Experience Relay

Initialize the replay buffer R;
Build a Q-network with random θ;
for *k from 1 to M* **do**
  state X_0 is set that no fog node is assigned;
  for *n from 1 to N* **do**
    $p_\epsilon \leftarrow random(0,1)$;
    if $p_\epsilon \leq 1-\epsilon$ **then**
      $A_n \leftarrow argmax_A Q(X_n, A_n; \theta_n)$;
    else
      A_n is randomly selected from set $\mathbb{A}_n$;
    end
    Observe reward r_n and assigned fog nodes from time $n+1$ to $n+1+max(n^a)$ by executing A_n;
    Set $X_{n+1} \leftarrow \{F - F^a_{n+1}, F - F^a_{n+2}, .., F - F^a_{n+1+max(n^a)}\}$;
    Store transition (X_n, A_n, r_n, X_{n+1}) in R;
    Sample transitions B as mini-batch from R;
    for *i from 1 to* $|B|$ **do**
      if X_{i+1} *is the terminal state* **then**
        $Y_i \leftarrow r_i$;
      else
        $Y_i \leftarrow r_i + \gamma \cdot \max_{A_{i+1}} Q(X_{i+1}, A_{i+1}; \theta_{n-1})$;
      end
      Update θ_n by performing the SGD;
    end
    $\theta_{n-1} \leftarrow \theta_n$
  end
end

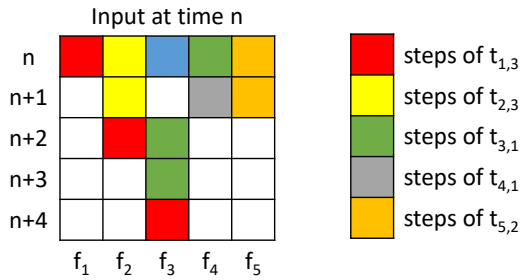


Fig. 4. Input structure of the state from time n to $n+4$ for the CNN network

The four input images are generated by four different actions denoted by A^1_n, A^2_n, A^3_n and A^4_n, respectively. The first convolutional layer uses 8×8 filters with stride 4 for the input images. There is also a nonlinear rectifier following the convolutional layer. The second convolutional layer uses 4×4 filters with stride 2, followed by a nonlinear rectifier. The third layer is a general fully connected hidden layer consisting of 256 rectifier units. The output layer is also a fully connected layer which outputs a single valid action. A difficulty is the number of tasks beginning at time n. If there are multiple tasks beginning at the same time,

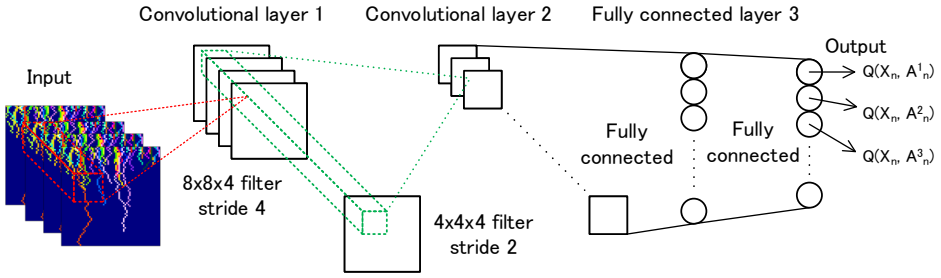


Fig. 5. Deep learning network structure used in the deep reinforcement scheduling

the number of valid actions will be much larger than the number of possible outputs of the neural network. In following experiments, we assume there is maximum 1 task beginning at each time slot for scheduling. Since the number of layers in most modern deep neural networks is no less than 100 and the traffic is reduced in the first several layers, we assume the number of steps is no more than 3. In task scheduling, the assigned fog nodes are limited to the nearby nodes in 1 hops. Therefore, the number of valid actions outputted by the last layer is no more than 9.

We then discuss the time complexity of the proposed algorithm. For a single convolutional layer m, the time complexity denoted by $\mathbf{O}_m$ is given by $O(\mathbf{M}^2 \cdot \mathbf{K}^2 \cdot \mathbf{C}_{in} \cdot \mathbf{C}_{out})$ where $\mathbf{M}$ is the side length of the feature map, $\mathbf{K}$ is the side length of a kernel, $\mathbf{C}_{in}$ is the number of input channels and $\mathbf{C}_{out}$ is the number of output channels. For the CCN model, the time complex is $O(\sum_{l=1}^{\mathbf{D}} \mathbf{O}_m)$. Since we use a four layer CCN model, the time complex is $O(\mathbf{X}^4)$ where X is the side length of the input matrix. Thus, the time complexity for scheduling one task is $O(|F|^4)$ since the size of the state matrix is $|F| \times |F|$.

5 PERFORMANCE EVALUATION

In this section, we first introduce the experiment settings and then analyze the experiment results.

5.1 Experiment Settings

We take extensive simulations to evaluate our solutions. In all simulation, we build a simulator with the Python 2.7 and networkx 1.6 on a workstation computer. The workstation computer has an Intel Core™i7 4770 (8MB cache, up to 3.90GHz) CPU, 16GB memory, a 128GB SSD and a 2TB hard-disk. The scheduler is developed in a server equipping an Intel Core™i7 7700 (8MB cache, up to 4.2GHz) CPU, 32GB memory, a 256GB SSD, a 3TB hard-disk and a NVIDIA GeForce GTX 1080 (8GB Memory) graphics card. We installed Ubuntu 16.04.3 LTS as the operating system and Keras 2.0.6 as the neural networks API. In all experiments, we use the RMSProp optimizer provided by Keras. The size of mini-batches is set to 32. The value of ϵ for the ϵ-greedy in Algorithm 1 is set to 0.1. The deep learning network is trained 1 million times. All input data are stored in the replay memory.

In all simulations, we use a tracing dataset from CRAWDAD, which records the activities of 100 mobile devices at Massachusetts Institute of Technology (MIT) in nine months [15]. Thus, the number of users is set to 100, and the position of each user is decided with the tracing data.

(a) AT&T Mobility

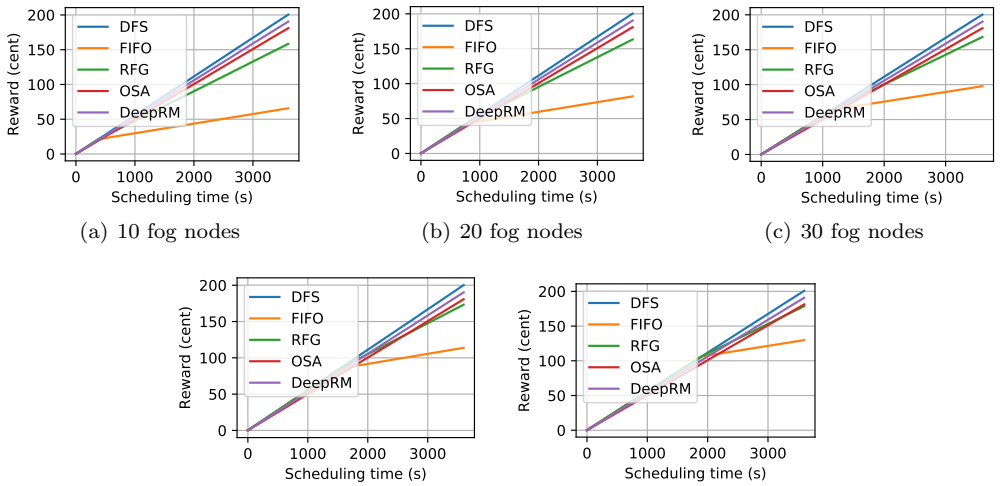
(b) T-mobile

Fig. 6. Cell towers near to MIT campus

As shown in Fig 6, we also investigate the positions of cell towers near to MIT campus from Cellmapper, including towers of AT&T Mobility and T-mobile. Since there are five cell towers from AT&T in Fig. 6(a) and five cell towers from T-mobile in Fig. 6(b), fog nodes in simulations are distributed in 10 positions near to these cell towers. The number of fog nodes is set from 10 to 50, while the number of fog nodes in the same position is from one to five. The mobile users connect the nearest cell towers geographically. The bandwidth between devices and fog nodes is distributed in [10, 35]Mbps for T-mobile, and [8, 32]Mbps for AT&T Mobility.

The bandwidth between fog nodes from the same company is set to 10Gbps with 10ms latency. The bandwidth between fog nodes and cloud servers is set to 1Gbps with 200ms latency. Between fog nodes from different companies, the bandwidth is set to 500Mbps with 400ms latency.

We also test the upload data of some mobile crowdsensing applications and set the size of the upload data of each task is distributed in [1, 10]MB and the size of the output data is set to 1KB of each task. Each user randomly requires an upload task for the mobile crowdsensing. The time interval of two tasks from the same user is distributed in [2, 4], [4, 6], [6, 8], [8, 10]seconds. The required TTC is set from one to five seconds, which is near to the average page loading time.



(a) 10 fog nodes (b) 20 fog nodes (c) 30 fog nodes

(d) 40 fog nodes (e) 50 fog nodes

Fig. 7. Rewards with different numbers of fog node in the first hour

The price of the output data is set to 1.54e−5dollar/KB. After each step, the data size is reduced 80 %. The computing cost of the cloud server is set to 0.003dollar per second from the price of the Amazon EC2 p3.8xlarge instance. The cost of uploading traffic is set to 2e−7dollar/KB from the price of dedicated Internet access service.

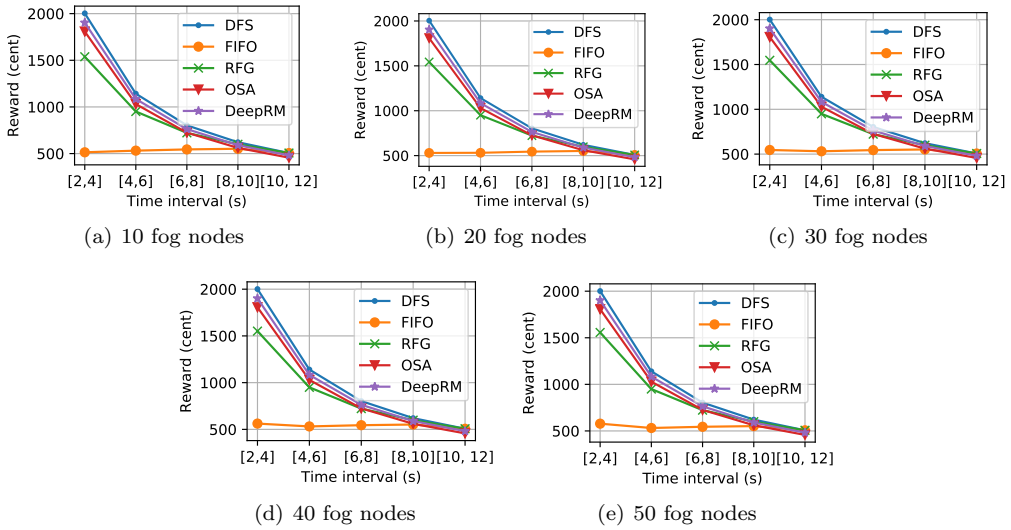
The length of a time slot is set to 100ms, and each task has three steps. The TTC of each step is distributed from [100, 300]ms.

We also compare the performance of our work named deep reinforcement fog scheduling (DFS) to following solutions.

- FIFO: the first in, first out strategy assigns the nearest fog nodes to tasks until there is no available fog node. After that, the strategy sends all tasks to the cloud server.
- RFG: the reward first greedy strategy assigns fog nodes to tasks for maximizing reward r_n at time n.
- OSA: the online scheduling algorithm assigns fog nodes to those tasks if the reward becomes more than a threshold value. If the reward is lower than the threshold value, the strategy submits the task to the cloud server.
- DeepRM: a resource management method with deep reinforcement learning [30]. We modify the deep neural network of DeepRM for scheduling tasks in fog computing. We use the same discount factor $\gamma = 0.9$ in the DeepRM as well as in our solution.

5.2 Numerical Results

For five different numbers of fog nodes in the simulation, we train five learning networks for these settings. We test the rewards with different fog nodes. After training all learning networks, we select one hour from the validation dataset as an example shown in Fig. 7. The time interval of two tasks from the same user is distributed in [2,4] seconds. Our solution performs better than other strategies with different numbers of fog nodes. Increased fog nodes improves the performance of FIFO and RFG. From the results shown in Fig. 7(a), the reward increasing rates of FIFO and RFG decrease after 500 seconds simulation with 10 fog



(a) 10 fog nodes (b) 20 fog nodes (c) 30 fog nodes

(d) 40 fog nodes (e) 50 fog nodes

Fig. 8. Rewards with different frequencies of crowdsensing task requiring in ten hours

nodes. When the number of fog nodes is increased to 20, the performance of FIFO and RFG decreases after 900 seconds as shown in Fig. 7(b). When we increase the number of fog nodes to 50, the reward increasing rates of FIFO and RFG decrease after 2000 seconds. As shown in Fig. 7(e), RFG performs better than OSA before 3000 seconds with 50 fog nodes in the simulation. Since FIFO and RFG assign as many fog nodes as possible to the tasks in online scheduling, the performance will decrease quickly when there are not enough available fog nodes. Since OSA reserves available fog nodes for the future tasks, there are always enough fog nodes for future assignment. Due to the reinforcement scheduling can learn the optimal decision for the future assignment after training, the performance of DFS and DeepRM is better in the fog computing. Since we apply CNN instead of the hidden-layer-based deep neural network in DeepRM, the performance of DFS is better than DeepRM in one-hour experiments.

We also test the performance of four strategies in a longer period. We test the rewards with different time intervals of two tasks from the same user. From the validation dataset, we choose the user mobility records from 9:00 a.m. to 19 a.m. We execute each simulation 20 times and record the average value of the rewards. As shown in Fig. 8, we find the frequencies of task requests obviously affect the performance of the scheduling. From the results in Fig. 8(a), the rewards decrease with longer intervals between task because the total amount of tasks is decreased. When users upload data in a high frequency, the performance of DFS and DeepRM is better than other solutions. The reward of DFS in 10 hours becomes more than 2000 cents which is higher than the reward of DeepRM. The number of fog nodes only affect slightly the performance of FIFO while other solutions perform similarly with different numbers of fog nodes. The reward of FIFO increases with more fog nodes in the experiment. The reward of FIFO is near to 500 cents with 10 fog nodes when the time interval is distributed from two to four seconds. When the number of fog nodes increases to 50, the reward of FIFO is more than 550 cents. However, the number of fog nodes makes a very small influence on the task scheduling in a long period.

As a result, the deep reinforcement scheduling shows promise for scheduling mobile crowdsensing tasks in fog computing. Since the fog node brings an additional cost to the service provider, it is hard to deploy enough fog nodes in access networks. For this issue, our solution shows good performance with limited fog nodes based on real-time task scheduling. Moreover, our solution can provide enough performance for processing user data with a high data collection frequency in mobile crowdsensing.

6 CONCLUSION AND FEATURE WORK

This paper shows that it is possible to apply deep reinforcement learning concept to schedule mobile crowdsensing tasks in fog computing. The results from extensive simulations show that the deep reinforcement scheduling is better than traditional solutions for a given fog computing environment. We plan to implement the strategy in a practical context that learning task scheduling decisions directly from experiences can offer a good opportunity for the future fog computing system.

ACKNOWLEDGMENTS

This work is supported by JSPS KAKENHI Grant Number JP16K00117, JP15K15976, JP17K12669, and Research Fund for Postdoctoral Program of Muroran Institute of Technology.

REFERENCES

[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*. USENIX Association, Berkeley, CA, USA, 19–19. http://dl.acm.org/citation.cfm?id=1855711.1855730

[2] M. A. Alsheikh, D. Niyato, S. Lin, H. p. Tan, and Z. Han. 2016. Mobile big data analytics using deep learning and apache spark. *IEEE Network* 30, 3 (May 2016), 22–29. https://doi.org/10.1109/MNET.2016.7474340

[3] S. Arai, K. Sycara, and T. R. Payne. 2000. Multi-agent reinforcement learning for planning and scheduling multiple goals. In *Proceedings Fourth International Conference on MultiAgent Systems*. 359–360. https://doi.org/10.1109/ICMAS.2000.858474

[4] R. Atallah, C. Assi, and M. Khabbaz. 2017. Deep reinforcement learning-based scheduling for roadside communication networks. In *2017 15th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*. 1–8. https://doi.org/10.23919/WIOPT.2017.7959912

[5] S. Basudan, X. Lin, and K. Sankaranarayanan. 2017. A Privacy-Preserving Vehicular Crowdsensing-Based Road Surface Condition Monitoring System Using Fog Computing. *IEEE Internet of Things Journal* 4, 3 (June 2017), 772–782. https://doi.org/10.1109/JIOT.2017.2666783

[6] Richard Bellman. 2013. *Dynamic programming*. Courier Corporation.

[7] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan. 2012. Bi-criteria Workflow Tasks Allocation and Scheduling in Cloud Computing Environments. In *2012 IEEE Fifth International Conference on Cloud Computing*. 638–645. https://doi.org/10.1109/CLOUD.2012.83

[8] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar. 2017. Mobility-Aware Application Scheduling in Fog Computing. *IEEE Cloud Computing* 4, 2 (March 2017), 26–35. https://doi.org/10.1109/MCC.2017.27

[9] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. 2014. *Fog Computing: A Platform for Internet of Things and Analytics*. Springer International Publishing, Cham, 169–186. https://doi.org/10.1007/978-3-319-05029-4_7

[10] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC '12)*. ACM, New York, NY, USA, 13–16. https://doi.org/10.1145/2342509.2342513

[11] D. Bruneo, S. Distefano, F. Longo, G. Merlino, A. Puliafito, V. D'Amico, M. Sapienza, and G. Torrisi. 2016. Stack4Things as a fog computing platform for Smart City applications. In *2016 IEEE Conference*

on Computer Communications Workshops (INFOCOM WKSHPS). 848–853. https://doi.org/10.1109/INFCOMW.2016.7562195

[12] Shin-Ming Cheng, Phone Lin, Di-Wei Huang, and Shun-Ren Yang. 2006. A Study on Distributed/Centralized Scheduling for Wireless Mesh Network. In *Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing (IWCMC '06)*. ACM, New York, NY, USA, 599–604. https://doi.org/10.1145/1143549.1143668

[13] Yohan Chon, Nicholas D. Lane, Fan Li, Hojung Cha, and Feng Zhao. 2012. Automatically Characterizing Places with Opportunistic Crowdsensing Using Smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp '12)*. ACM, New York, NY, USA, 481–490. https://doi.org/10.1145/2370216.2370288

[14] Salvatore Distefano, Francesco Longo, and Marco Scarpa. 2015. QoS Assessment of Mobile Crowdsensing Services. *Journal of Grid Computing* 13, 4 (01 Dec 2015), 629–650. https://doi.org/10.1007/s10723-015-9338-7

[15] Nathan Eagle and Alex (Sandy) Pentland. 2005. CRAWDAD dataset mit/reality (v. 2005-07-01). Downloaded from https://crawdad.org/mit/reality/20050701. (July 2005). https://doi.org/10.15783/C71S31

[16] Eyal Even-Dar and Yishay Mansour. 2001. Learning Rates for Q-Learning. In *Computational Learning Theory*, David Helmbold and Bob Williamson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 589–604.

[17] R. K. Ganti, F. Ye, and H. Lei. 2011. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine* 49, 11 (November 2011), 32–39. https://doi.org/10.1109/MCOM.2011.6069707

[18] Robert Glaubius, Terry Tidwell, Christopher Gill, and William D. Smart. 2010. Real-time Scheduling via Reinforcement Learning. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence (UAI'10)*. AUAI Press, Arlington, Virginia, United States, 201–209. http://dl.acm.org/citation.cfm?id=3023549.3023573

[19] D. Hoang and T. D. Dang. 2017. FBRC: Optimization of task Scheduling in Fog-Based Region and Cloud. In *2017 IEEE Trustcom/BigDataSE/ICESS*. 1109–1114. https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.360

[20] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. 2013. Mobile Fog: A Programming Model for Large-scale Applications on the Internet of Things. In *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing (MCC '13)*. ACM, New York, NY, USA, 15–20. https://doi.org/10.1145/2491266.2491270

[21] H. Hu, Y. Wen, T. S. Chua, J. Huang, W. Zhu, and X. Li. 2016. Joint Content Replication and Request Routing for Social Video Distribution Over Cloud CDN: A Community Clustering Method. *IEEE Transactions on Circuits and Systems for Video Technology* 26, 7 (July 2016), 1320–1333. https://doi.org/10.1109/TCSVT.2015.2455712

[22] H. Hu, Y. Wen, and D. Niyato. 2017. Public Cloud Storage-Assisted Mobile Social Video Sharing: A Supermodular Game Approach. *IEEE Journal on Selected Areas in Communications* 35, 3 (March 2017), 545–556. https://doi.org/10.1109/JSAC.2017.2659478

[23] H. Hu, Y. Wen, and D. Niyato. 2017. Spectrum Allocation and Bitrate Adjustment for Mobile Social Video Sharing: Potential Game With Online QoS Learning Approach. *IEEE Journal on Selected Areas in Communications* 35, 4 (April 2017), 935–948. https://doi.org/10.1109/JSAC.2017.2676598

[24] S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang. 2012. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *2012 Proceedings IEEE INFOCOM*. 945–953. https://doi.org/10.1109/INFCOM.2012.6195845

[25] H. Li, K. Ota, M. Dong, and M. Guo. 2017. Mobile Crowdsensing in Software Defined Opportunistic Networks. *IEEE Communications Magazine* 55, 6 (2017), 140–145. https://doi.org/10.1109/MCOM.2017.1600719

[26] H. Li, K. Ota, M. Dong, A. Vasilakos, and K. Nagano. 2017. Multimedia Processing Pricing Strategy in GPU-accelerated Cloud Computing. *IEEE Transactions on Cloud Computing* PP, 99 (2017), 1–1. https://doi.org/10.1109/TCC.2017.2672554

[27] L. Li, K. Ota, M. Dong, and W. Borjigin. 2017. Eyes in the Dark: Distributed Scene Understanding for Disaster Management. *IEEE Transactions on Parallel and Distributed Systems* 28, 12 (Dec 2017), 3458–3471. https://doi.org/10.1109/TPDS.2017.2740294

[28] T. Li, Y. Liu, L. Gao, and A. Liu. 2017. A Cooperative-Based Model for Smart-Sensing Tasks in Fog Computing. *IEEE Access* 5 (2017), 21296–21311. https://doi.org/10.1109/ACCESS.2017.2756826

[29] Yuanyuan Li, Zhiyang Li, Mianxiong Dong, Wenyu Qu, Changqing Ji, and Junfeng Wu. 2015. Efficient subspace skyline query based on user preference using MapReduce. *Ad Hoc Networks* 35, Supplement C (2015), 105 – 115. https://doi.org/10.1016/j.adhoc.2015.07.006 Special Issue on Big Data Inspired Data Sensing, Processing and Networking Technologies.

[30] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. ACM, New York, NY, USA, 50–56. https://doi.org/10.1145/3005745.3005750

[31] Giovanni Merlino, Stamatis Arkoulis, Salvatore Distefano, Chrysa Papagianni, Antonio Puliafito, and Symeon Papavassiliou. 2016. Mobile crowdsensing as a service: A platform for applications on top of sensing Clouds. *Future Generation Computer Systems* 56, Supplement C (2016), 623 – 639. https://doi.org/10.1016/j.future.2015.09.017

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:1312.5602 http://arxiv.org/abs/1312.5602

[33] M. Mohammadi, A. Al-Fuqaha, M. Guizani, and J. S. Oh. 2018. Semisupervised Deep Reinforcement Learning in Support of IoT and Smart City Services. *IEEE Internet of Things Journal* 5, 2 (April 2018), 624–635. https://doi.org/10.1109/JIOT.2017.2712560

[34] Xuan-Qui Pham and Eui-Nam Huh. 2016. Towards task scheduling in a cloud-fog computing system. In *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 1–4. https://doi.org/10.1109/APNOMS.2016.7737240

[35] Mahadev Satyanarayanan. 2010. Mobile Computing: The Next Decade. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond (MCS '10)*. ACM, New York, NY, USA, Article 5, 6 pages. https://doi.org/10.1145/1810931.1810936

[36] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos. 2015. Edge Analytics in the Internet of Things. *IEEE Pervasive Computing* 14, 2 (Apr 2015), 24–31. https://doi.org/10.1109/MPRV.2015.32

[37] S. K. Sharma and X. Wang. 2017. Live Data Analytics With Collaborative Edge and Cloud Processing in Wireless IoT Networks. *IEEE Access* 5 (2017), 4621–4635. https://doi.org/10.1109/ACCESS.2017.2682640

[38] W. Sherchan, P. P. Jayaraman, S. Krishnaswamy, A. Zaslavsky, S. Loke, and A. Sinha. 2012. Using On-the-Move Mining for Mobile Crowdsensing. In *2012 IEEE 13th International Conference on Mobile Data Management*. 115–124. https://doi.org/10.1109/MDM.2012.58

[39] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550 (Oct. 2017), 354. http://dx.doi.org/10.1038/nature24270

[40] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556 http://arxiv.org/abs/1409.1556

[41] Karolj Skala, Davor Davidovic, Enis Afgan, Ivan Sovic, and Zorislav Sojat. 2015. Scalable Distributed Computing Hierarchy: Cloud, Fog and Dew Computing. *Open Journal of Cloud Computing (OJCC)* 2, 1 (2015), 16–24.

[42] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang. 2018. Machine Learning for Networking: Workflow, Advances and Opportunities. *IEEE Network* 32, 2 (March 2018), 92–99. https://doi.org/10.1109/MNET.2017.1700200

[43] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (01 May 1992), 279–292. https://doi.org/10.1007/BF00992698

[44] L. Xiao, Y. Li, G. Han, H. Dai, and H. V. Poor. 2018. A Secure Mobile Crowdsensing Game With Deep Reinforcement Learning. *IEEE Transactions on Information Forensics and Security* 13, 1 (Jan 2018), 35–47. https://doi.org/10.1109/TIFS.2017.2737968

[45] L. Xiao, X. Wan, C. Dai, X. Du, X. Chen, and M. Guizani. 2018. Security in Mobile Edge Caching with Reinforcement Learning. *ArXiv e-prints* (Jan. 2018). arXiv:cs.CR/1801.05915

[46] Yu Xiao, Pieter Simoens, Padmanabhan Pillai, Kiryong Ha, and Mahadev Satyanarayanan. 2013. Lowering the Barriers to Large-scale Mobile Crowdsensing. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications (HotMobile '13)*. ACM, New York, NY, USA, Article 9, 6 pages. https://doi.org/10.1145/2444776.2444789

[47] Shanhe Yi, Cheng Li, and Qun Li. 2015. A Survey of Fog Computing: Concepts, Applications and Issues. In *Proceedings of the 2015 Workshop on Mobile Big Data (Mobidata '15)*. ACM, New York,

NY, USA, 37–42. https://doi.org/10.1145/2757384.2757397

[48] Wei Zhang and Thomas G. Dietterich. 1995. A Reinforcement Learning Approach to Job-shop Scheduling. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2 (IJCAI'95)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1114–1120. http://dl.acm.org/citation.cfm?id=1643031.1643044

[49] D. Zhao, X. Y. Li, and H. Ma. 2014. How to crowdsource tasks truthfully without sacrificing utility: Online incentive mechanisms with budget constraint. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 1213–1221. https://doi.org/10.1109/INFOCOM.2014.6848053

[50] Z. Zheng, L. X. Cai, M. Dong, X. Shen, and H. V. Poor. 2011. Constrained Energy-Aware AP Placement with Rate Adaptation in WLAN Mesh Networks. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*. 1–5. https://doi.org/10.1109/GLOCOM.2011.6134158