Resource Management at the Network Edge: A Deep Reinforcement Learning Approach

Deze Zeng, Lin Gu, Shengli Pan, Jingjing Cai, and Song Guo

Abstract

With the advent of edge computing, it is highly recommended to extend some cloud services to the network edge such that the services can be provisioned in the proximity of end users, with better performance efficiency and cost efficiency. Compared to cloud computing, edge computing has high dynamics, and therefore the resources shall be correspondingly managed in an adaptive way. Traditional model-based resource management approaches are limited in practical application due to the involvement of some assumptions or prerequisites. We think it is desirable to introduce a model-free approach that can fit the network dynamics well without any prior knowledge. To this end, we introduce a model-free DRL approach to efficiently manage the resources at the network edge. Following the design principle of DRL, we design and implement a mobility-aware data processing service migration management agent. The experiments show that our agent can automatically learn the user mobility pattern and accordingly control the service migration among the edge servers to minimize the operational cost at runtime. Some potential future research challenges are also presented.

Introduction

Edge computing, as a newly emerging computing paradigm, provides an alternative and promising platform to process tasks by exploring various resources at the network edge. Unlike centralized cloud, edge servers (e.g., servers with cellular base stations) are geo-distributed in different locations, naturally enabling performance and cost efficiency improvement by processing data close to their sources. As indicated in [1], it is highly recommended to host latency-sensitive services, usually in the form of virtual machines (VMs), at the network edge such that the data can be processed as soon as they are generated, instead of transferring all of them to the distant cloud with considerable communication overhead.

When talking about providing services in edge computing, the biggest challenge is that the edge exhibits high dynamics in both resource provision and resource demand. For example, the communication and computation resources in the network edge may vary in time as the resources are shared by different applications in the edge. Basically, the tasks and users participating in an edge hosted application usually move within the network and may associate with different edge servers at different time, incurring time-varying communication and computation resource requirements. It is essential to handle the edge resource management in a dynamic way by adapting to such network dynamics. Actually, this phenomenon has already come to the notice of researchers, and many dynamic resource management strategies capable of dealing with the network dynamics have been proposed. For example, Chen *et al.* [2] study the task offloading problem in mobile edge computing considering energy dynamics in the battery of mobile devices. Mao *et al.* [3] apply the Lyapunov optimization algorithm and develop a computation offloading strategy to adapt to green energy dynamics for edge computing with energy harvesting devices. We notice that existing studies are mostly model-based with certain assumptions or prerequisites, hindering the actual deployment and application in practice.

Recently, there was big news that DeepMind's AlphaGo defeated the legendary Lee Sedol, who is one of the world's top Go players. It is regarded as a new milestone of artificial intelligence (AI) and ignites the recent intensity in the research on AI. The core technique in the design of AlphaGo, and its improved version AlphaGo Zero, is deep reinforcement learning (DRL), which is an improved version of reinforcement learning (RL) with the integration of deep learning. Once well trained, an RL/DRL agent is able to perform appropriate control action (e.g., position selection) based on the real-time state of the environment (e.g., Go board) to pursue the predefined objective (e.g., winning the game). The power of RL, as well as DRL, has been witnessed in a diversity of domains with complex control problems, including robotics, games, manufacturing, financial, and so on. Some recent studies also introduce it into the management and control problem in information systems, for example, wireless network management [4], energy management [5, 6] and mobile data offloading [7], TCP contention window size setting [8], and flow scheduling [9].

As a result, we are naturally motivated to apply it to manage the resources at the network edge according to the network dynamics toward any desired objective. In this article, we first briefly review the core concepts of RL and DRL, as well as their applications in the following section. We then present an RL-based edge computing resource management framework. Based on the framework, we study the problem of mobili-

Digital Object Identifier:
10.1109/MNET.2019.1800386

Deze Zeng and Shengli Pan are with China University of Geosciences;
Lin Gu (corresponding author) and Jingjing Cai are with Huazhong University of Science and Technology; Song Guo is with the Hong Kong Polytechnic University.

ty-aware service migration and energy control in edge computing and design a DRL-based algorithm to deal with the network dynamics, aiming at operational cost minimization. Extensive experiments are conducted to verify the efficiency of our strategy. We also intensively discuss the findings observed from the experiment results to give insight on why DRL is beneficial to edge resource management. Thereafter, we discuss the limitations and future potential research challenges on applying RL/DRL for edge computing resource management. Finally, we conclude our work.

RL and Its Applications

A Brief Understanding of RL

RL is an important branch of machine learning and usually focuses on solving control problems with delayed rewards over time. Different from traditional machine learning algorithms requiring large historical datasets, RL can start from scratch and gradually achieve human-level control capability. The key idea of RL is a smart agent receiving environment states and making ideal actions based on historical experience. There are a variety of RL algorithms such as Q-learning (and its variants, Sarsa and Sarsa(λ)), policy gradients, and actor critic.

Any RL algorithm consists of two phases: the *training* phase and the *inference* phase. The training phase is to tell the agent which action shall be taken under a given environment from a series of trials. Imagine a baby agent is to control the edge computing platform (*environment*) toward the goal of improving the user experience (*reward*). The baby agent will first look around and construct its own representation of the environment as the *state*, for example, current available resource amount, user demands, and service latency. Sadly, the curious baby agent has no knowledge about what to do and will start to explore the environment by making random decisions (*actions*). Then these actions will be carried out and applied to the network, like selecting the associated edge server, scheduling user tasks, and allocating edge resources. After these actions are taken, the baby agent will see how the network responds (next state). If the user experience is not good, the baby agent will feel sad and dislike it (receiving a negative reward) and will possibly not take these actions under the same or a similar situation in the future and vice versa. In this case, the agent has gained some experience. When an unexperienced state is observed, the agent can choose actions by exploiting existing experience or exploring a random decision. This procedure will repeat until the agent can always find a satisfactory solution to deal with different situations (i.e., with converged rewards).

After the training phase, the baby agent is grown up and can be applied to the inference phase, that is, taking appropriate actions according to the experience learned during the training.

A Review of RL Applications

The superiority of RL has been verified in many subareas of information systems. In wired networks, He *et al*. [10] propose an RL-based framework that enables dynamic orchestration of networking, caching, and computing resources to improve the performance of applications for smart cities. Maleki *et al.* [11] present a multi-objective routing protocol based on RL, aiming to lower the expected long-term energy computation and reduce the end-to-end delay. Chen *et al.* [9] invent a DRL-based automatic traffic optimization for data center traffic flows. In wireless networks, Wang *et al*. [12] investigate how to apply RL to the dynamic multi-channel access problem to maximize the expected long-term number of successful transmissions. Ortiz *et al.* [6] find an RL-based power allocation policy to maximize throughput and green energy utilization. Yan *et al.* [13] apply RL to guarantee the quality of service (QoS) of a heterogeneous network coexisting with cellular and WiFi, and to maximize the system throughput while meeting diverse traffic requirements. In computing, Ranadheera *et al*. [14] explore game theory and RL for efficient distributed resource management and task offloading decisions in mobile edge computing. Xu *et al.* [15] propose an RL-based task scheduling algorithm to reduce the energy consumption for big data processing in data centers.

The existing work reveals that RL-based management and control algorithms are model-free and adapt well to the network dynamics in the absence of network statistics.

As a model-free approach, our RL-based edge computing resource management framework does not require any prior knowledge on the network dynamics or statistics. It can automatically learn the network dynamics and make appropriate control decisions accordingly at runtime.

RL-Based Edge Resource Management Framework

The power and success of RL inspire us to introduce it into edge computing resource management.

Framework Overview

As a model-free approach, our RL-based edge computing resource management framework does not require any prior knowledge on the network dynamics or statistics. It can automatically learn the network dynamics and make appropriate control decisions accordingly at runtime. In this section, we provide a detailed illustration of the proposed RL-based resource management framework as shown in Fig. 1, which consists of three modules: *network hypervisor*, *RL-based controller*, and *action executor*.

The *network hypervisor* is responsible for monitoring and aggregating the states of the elements in the edge computing environment; to name just a few, user locations, server workloads, service placement statuses, link bandwidth consumption, electricity prices, spectrum allocation statuses, user association, and base station power statuses. The information is essential for an RL-based controller to profile the environment so as to make appropriate control decisions. The more information collected, the higher resolution of the environment "picture" can be obtained, potentially increasing the control accuracy and efficiency. Of course, this is not free. More information also indicates high control complexity to the RL-based

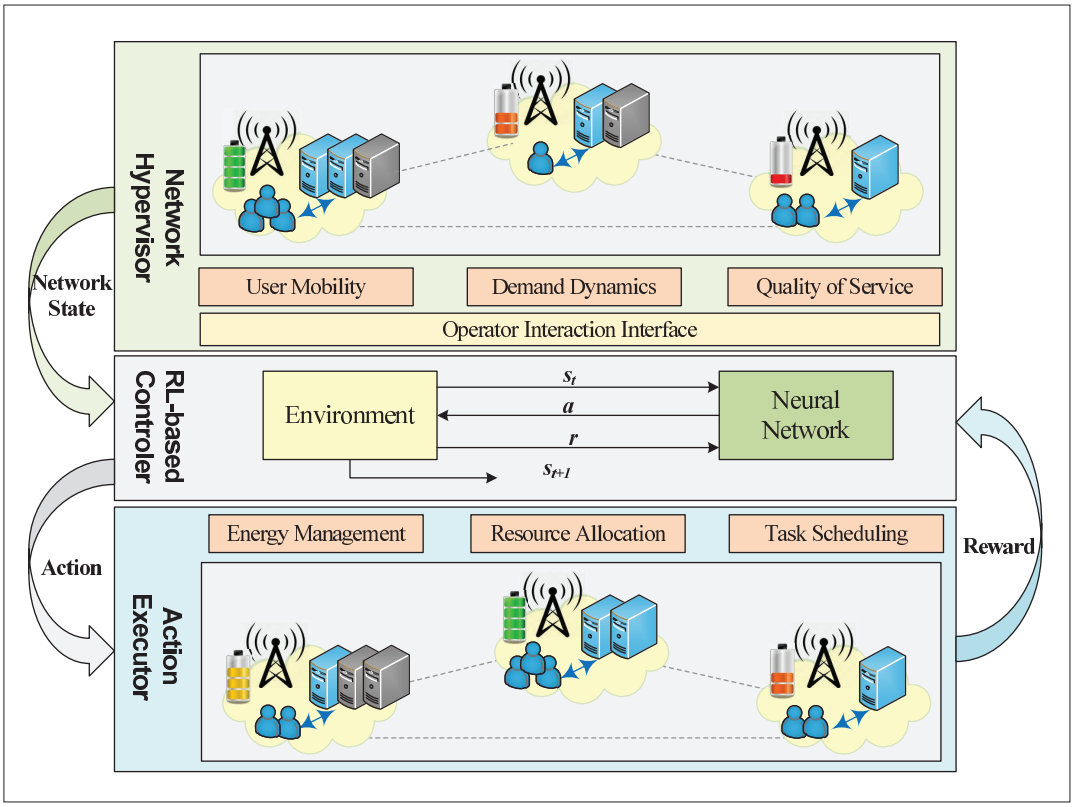

FIGURE 1. The RL-based edge resource management framework.

controller. The hypervisor also provides a programmable interface to the network operator for handling the interaction between the network operator and the RL-based controller.

The information collected by the network hypervisor is sent to the *RL-based controller* as the input to produce corresponding control actions as the output. The core of the RL-based controller is RL/DRL algorithms, and therefore it requires lots of data to train. We do not need to give the controller manually designed statistics but simply feed it with the data required. The controller can learn itself according to the objective we set (e.g., improving the QoS, lowering the operational expenditure [OPEX], reducing power consumption). The inherent characteristics of RL/DRL make the controller learn from the history and look into the future. Usually, it is impractical to train the controller online directly as an unsmart controller may generate inefficient control actions, which may lead to disastrous results. Therefore, we may apply historical data and simulator/emulator to train an early-version controller offline. Once well trained, the RL-based controller is capable of making decisions based on the real-time environment state and its accumulated experience. Note that the controller can always learn by updating itself at runtime. Therefore, the controller shall become smarter and smarter in use, not only in training.

The *action executor* on each controllable edge computing element (e.g., edge server, base-station, user equipment, router, and so on) can communicate with the RL-based controller to obtain the control decisions, and accordingly execute the derived action. Once the action is taken, the executor calculates the reward obtained on each network node and reports it back to the RL-based controller to update the control agent so as to make it more intelligent and efficient.

Selection of RL/DRL for the Controller

There are already a number of different RL/DRL algorithms with different characteristics available that can be applied to resource management in edge computing.

The very basic RL solution is called Q-learning, which chooses actions according to the Q-values stored in its two-dimensional Q-table. With Q-learning, we can solve some control problem with discrete actions, for example, service placement or the network flow routing problem in a small-scale environment. Although Q-learning is powerful and simple, its main weakness is the lack of generality and scalability. That is, as the states (the number of edge servers, the number of users, the number of edge services, etc.) rise, the Q-table may exponentially increase, which makes Q-learning inapplicable in large-scale networks. To tackle this issue, the deep Q network (DQN) introduces a neural network to estimate the Q-value function. Thanks to the neural network, DQN can be applied to the large-scale problem. It is noticeable that neither Q-learning nor DQN can cope with continuous problems. Hence, deep deterministic policy gradient (DDPG), as the representative advanced DRL technique, employs the actor-critic model and is capable of dealing with continuous control problems like workload balancing, task offloading, resource allocation, traffic optimization, and energy scheduling. When deciding which algorithms to apply to a specific problem, different RL/DRL solutions should be used carefully with respect to the algorithm characteristics and the problem requirements.

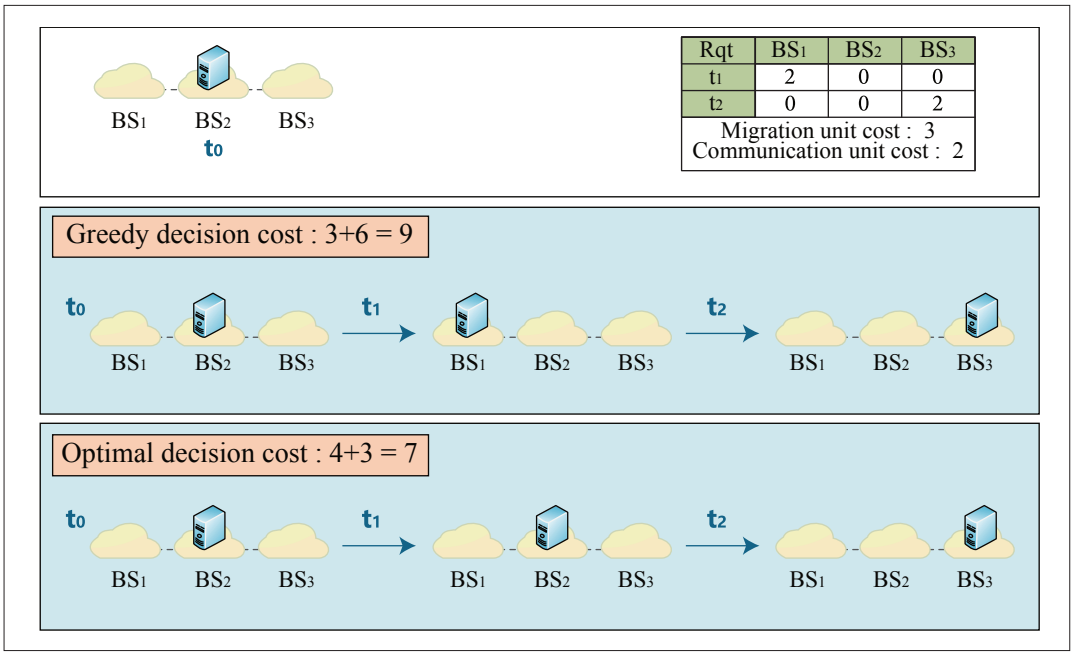

FIGURE 2. A toy example of the influence of VM migration on the overall operational cost.

Advantages of RL-Based Edge Resource Management

RL-based solutions have already shown great significance in resource management and control problems. They also naturally fit edge computing resource management thanks to the following distinctive advantages.

High Accuracy: RL-based solutions are data-driven. All the decisions are based on the profiling of the environment, without man-made assumptions or statistics. Fruitful raw information sometimes is helpful to make accurate control decisions.

Fast Decision: Although it requires lots of data and a long time to train an RL-based controller to become effective, once trained, it can make inferences quite fast, as witnessed in AlphaGo and AlphaZero.

Ever Evolving: Even after application, the RL controller can still learn by itself from the effects of actual control. It can self-adjust after facing new environments and accumulate new experiences to evolve.

Easy Implementation: RL-based solutions can easily be implemented and deployed as they do not require solving complicated mathematical models and simply require obtaining the data needed. Besides, there are already many RL/DRL frameworks available.

A Case Study: The DQN-Based Service Migration Approach in Edge Computing

In this section, we introduce a case study of applying our RL-based edge resource management framework to the specific problem of user-mobility-aware VM migration for operational cost minimization at runtime.

Problem Statement

Let us first use a toy example considering a simple edge computing platform with one VM and three edge servers colocated with cellular base stations, as shown in Fig. 2, to state the problem to be addressed. In this example, the VM provides a service shared by the mobile users. We are mainly interested in lowering the operational cost. The cost consists of two aspects: migration cost incurred by migrating VMs between edge servers, and communication cost between the VM and the users. The unit data communication cost and VM migration cost are 2 and 3, respectively. Starting from time slot t_0, the VM is initially placed on BS_2. At time t_1, two users are associated with BS_1 and then move to BS_3 at time t_2. If we optimize the overall cost on each time slot, we shall migrate the VM to BS_1 with migration cost 3, and no data communication cost is incurred. Then the VM is also moved to BS_3 with migration cost 6. Similarly, the data communication cost is 0. In this case, we achieve a total cost of 9 till t_2. It can easily be derived that the optimal solution is to keep the VM on BS_2 at time t_1, and then move it to node 3 at time t_2, achieving an overall cost of 7.

However, it is nontrival to obtain such an optimal solution at runtime without knowing the user association beforehand. Therefore, our problem is to decide the VM placement location (i.e., the base station) at runtime according to the user mobility dynamics so as to reduce the overall operational cost.

RL-Based Algorithm Design

According to the design principle of RL (including DRL), we shall first accurately define the *state space* $\mathcal{S}$, *action space* $\mathcal{A}$, and *reward* r, according to the problem to be solved. For the problem stated above, the core elements used in our RL-based algorithm are defined as follows.

$\mathcal{S}$: The state $s \in \mathcal{S}$ at time t is represented by a tuple consisting of the current location (or base station) of the VM and the number of users associated to each base station, that is,

$$s^t \triangleq \left(v^t, (u_1^t, u_2^t, u_3^t, \ldots, u_i^t, \ldots)\right),$$

1: Initialize the Q-Network with random weights θ
2: Initialize the replay memory $\mathcal{D}$ to capacity $\mathcal{K}$
3: **for all** available episodes **do**
4: With probability ε, pick a random action a^t at state s^t
5: With probability $(1-\varepsilon)$, take a^t that maximizes $\mathcal{Q}(s^t, a|\theta)$
6: Take the action a^t to obtain reward $r(s^t, a^t)$ and observe s^{t+1}
7: Store (s^t, a^t, r^t, s^{t+1}) in $\mathcal{D}$
8: Choose random mini-batch transitions, i.e., sampling $\{(s^i, a^i, r^i, s^{i+1})\}$ from $\mathcal{D}$
9: $\forall$ (s^i, a^i, r^i, s^{i+1}) in the mini-batch transitions, set the predicted reward $\eta^i = r^i$ if the current episode ends at step $i+1$; otherwise, set $\eta^i = r^i + \gamma \max_{a'} \mathcal{Q}(s^{i+1}, a')$
10: Conduct gradient descent with loss given by $(\eta^i - \mathcal{Q}(s^i, a^i|\theta))^2$
11: **end for**

ALGORITHM 1. DQN-based VM migration control.

Hence, the state space $\mathcal{S}$ consists of all the possible combinations of v and (u_1, u_2, u_3, ..., u_i, ...).

$\mathcal{A}$: Suppose that the VM is located in $v^t = BS_i$ at time t, and it can migrate to $v^{t+1} = BS_j$. We can simply regard the action a^t as "move to BS_j," denoted by $a^t \triangleq BS_j$, Hence, we have $\mathcal{A} = \{BS_k | k = 1, 2, 3, ...\}$ as the VM placement candidate set.

r: The agent, after taking action a^t on state s^t at time t, shall receive a reward as $r(s^t, a^t) = 1/(W(s^t, a^t) + M(s^t, a^t))$, where $W(s^t, a^t)$ and $M(s^t, a^t)$ are the communication cost for the data transmission and VM migration during time slot t, respectively. Obviously, maximizing the reward is equivalent to minimizing the overall cost.

Based on the above definitions, we can start the design of the training phase of our algorithm. Note that in the inference phase, the trained agent simply takes the real-time state as input to obtain the decision on the VM migration as output. Therefore, the inference phase is omitted for brevity, and we mainly focus on the training phase, which actually is the most essential part of any RL-based algorithm. Unfortunately, in practice, the curse of dimensions will emerge during the training phase due to the large size of state space. This was actually also observed during our initial trials on applying Q-learning in various network sizes (i.e., the number of base stations in this problem). When the network size is large, it generates an extremely large state space, making it hard for the agent to converge during the training phase. As an efficient way to tackle this challenge, we employ a neural network (i.e., Q-network) to represent the Q-function (i.e., the state-action value function), rather than recording them individually in the form of a Q-table. Details of our DQN-based VM migration control training algorithm are presented in Algorithm 1.

We first randomly initialize the weights θ of the Q-network that is implemented by a deep neural network to approximate the Q-function used in the framework of DQN in line 1. Specifically, an action a^t taken in a state s^t (at time t) is related to the reward $r(s^t, a^t)$ received as a measurement of how good the action is. Q-function is then introduced to represent the long-term reward that can be obtained by control policy π, that is,

$$\mathcal{Q}^{\pi}(s^t, a^t) \triangleq \mathbb{E}\left[\sum_{k=t}^{T} \gamma^{k-t} r(s^t, a^t) \mid s^t, a^t, \pi\right], \quad (1)$$

where T represents the total time slots and $\gamma \in [0, 1]$ is the discount factor indicating the certainty of the agent in the future state transitions and actions to be taken. The optimal control policy π^* is the one that maximizes the above long-term reward, that is, $\mathcal{Q}^{\pi^*}(s^t, a^t) = \max_{\pi} \mathcal{Q}^{\pi}(s^t, a^t)$.

The Q-network takes state s as input and outputs $\mathcal{Q}(s, a|\theta)$ for every possible action a. It is a nonlinear function approximator and is known to make RL unstable or even diverge. To overcome this, experience replay, which randomizes over the data so as to remove correlations in the observation sequence and smooths over changes in the data distribution, is employed by DQN. Therefore, we initialize a replay memory $\mathcal{D}$ with size $\mathcal{K}$ in line 2. Then we try to find the Q-network that makes $\mathcal{Q}(s,a|\theta) \approx \mathcal{Q}^*(s,a)$ through an iteration way in lines 3–11.

We apply an ε-greedy approach to train the Q-network by exploring a random action with probability ε (line 4), and exploiting an action that maximizes the expected long-term reward with probability $1 - \varepsilon$ (line 5). For each action taken, we shall correspondingly obtain its new state and the resultant reward. This is treated as the experience in RL and shall be reserved for future use. Therefore, we store the experiences $(s^t, a^t, r(s^t, a^t), s^{t+1})$ in the relay memory $\mathcal{D}$ in line 7.

The training phase actually is the update of the weights θ^t of the Q-network. After obtaining an experience, we renew the values of θ^t with samples stored in the relay memory $\mathcal{D}$. Rather than by a single record, samples (or mini-batches) of experiences are drawn uniformly at random from the relay memory in line 8. Thereafter, we try to find the optimal θ^* that minimizes the loss gradient descent in line 10.

Experiments and Discussions

To verify the feasibility of our framework and the efficiency of our proposed DQN-based algorithm, we have conducted extensive simulation-based experiments. We initialize an edge computing platform consisting of 50 servers on different base stations in a randomly generated topology. The communication cost between any servers are set as the number of hops between them. There are 500 users who randomly move between these base stations. To check the capability of the model-free learning-based approach, we assume two different mobility models. Note that we do not give any information about the mobility model to our algorithm. The Q-network used in our algorithm is implemented with Tensorflow.

The Training Phase

Before applying the agent to actually control the VM migration, we shall first train the agent using Algorithm 1. To gain insight on the training process, we detail the loss, the reward, and the Q-value during the training process in Fig. 3. Figure 3a shows the loss value on different training steps. It can be observed that the loss gradually decreases and converges to 0 with increasing training steps. It indicates that the agent can be well trained to converge. Meanwhile, we depict the average

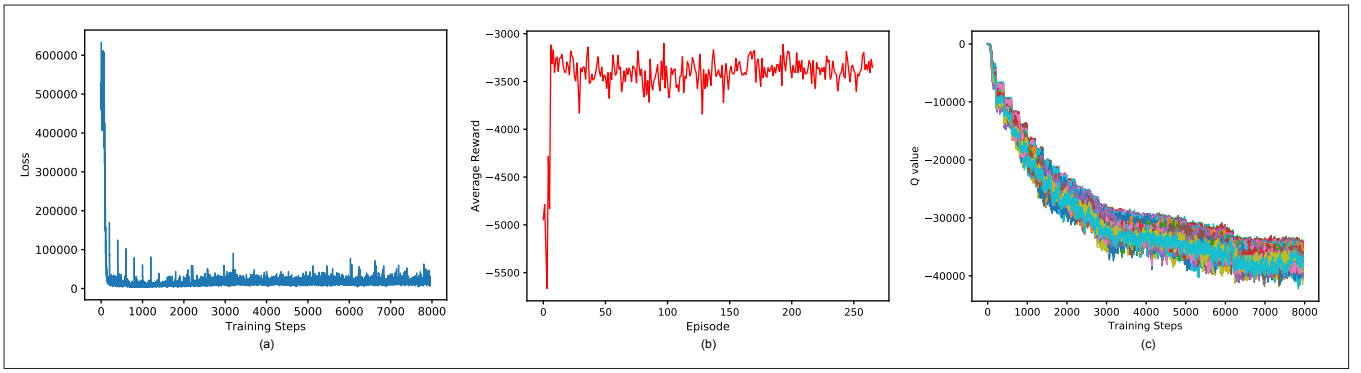

FIGURE 3. a) The loss; b) average reward; c) Q value during the training phase.

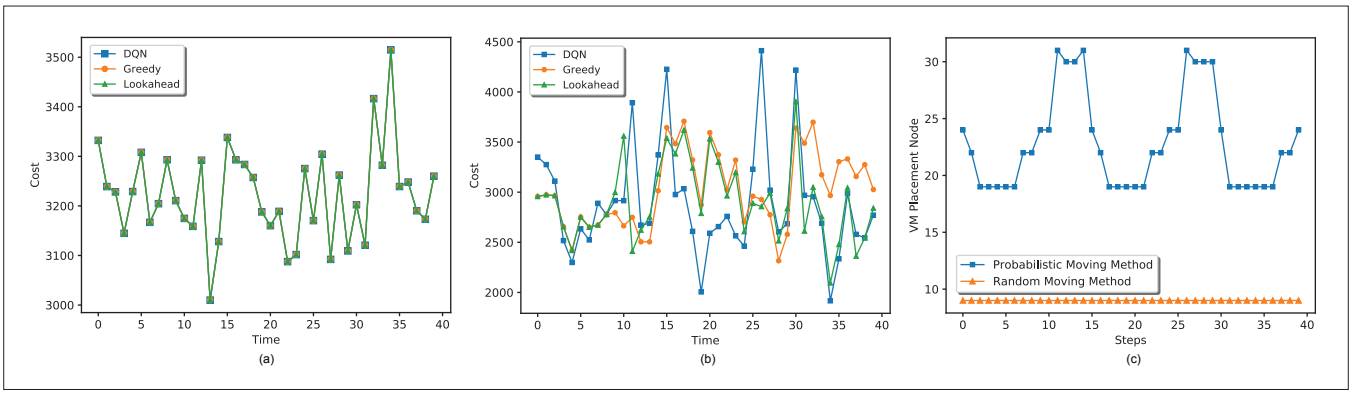

FIGURE 4. The performance evaluation results in the inference phase: a) the immediate cost in random movement; b) the immediate cost in probablistic movement; c) the VM placement location.

reward on each episode with 300 steps in Fig. 3b, from which we can observe that the average reward gradually increases and is basically stable after about 50 episodes. We further report the Q-values in Fig. 3c from which we notice that the Q-value also converges after about 5000 steps. At this point, we already obtain a well trained agent and can apply it in practical control.

The Inference Phase

Next, we use the trained network to assess the performance of our proposed DQN-based algorithm. Two different mobility models, probabilistic movement and random movement, are considered. In probabilistic movement, the mobile users transit between different base stations with preset probabilities, while they transit between different base stations with the same probability in random movement. We train the agent in different scenarios to evaluate the performance of our algorithm in the inference phase. Moreover, we compare the performance of our algorithm (DQN) with two competitors, Greedy and Lookahead. The Greedy algorithm is the one shown in our toy example that minimizes the total cost in each time slot without consideration of future conditions. The Lookahead algorithm is based on [2], which migrates the VM based on the prediction of the user movement so as to minimize the overall cost for 10 time slots.

We report the immediate cost in each time slot for 40 time slots in Fig. 4. Let us first check the performance in the scenario of random movement shown in Fig. 4a. We can see that the three lines are almost overlapped. In the random movement case, the performance of DQN is similar to the other two competitors. However, when it comes to probabilistic movement, the performance of the three algorithms deviates from each other, as shown in Fig. 4b. Although our algorithm does not always exhibit the best immediate cost efficiency, we notice that the average cost obtained by the three algorithms eventually is 2890.76, 3017.92, and 2907.07, respectively. Our DQN-based algorithm exhibits the best cost efficiency in the long run, achieving our design goal.

To further understand the performance results observed in Figs. 4a and 4b, we further elaborate the location of the VM placement of our DQN-based algorithm in each time slot in Fig. 4c. In random movement, we notice that the location of the VM almost does not change at runtime. The immobility of the VM is attributed to the fact that the users are dispersed in the network, and the number of users on each base station does not change too much in different time slots. Therefore, the VM shall always be located in the base station that can minimize the data transfer cost from the MCS data sources to the VM. Although our algorithm performs the same as the two other competitors, it proves that our algorithm indeed observes this phenomenon and can automatically make the right decision at runtime. This is further verified in the probabilistic movement case, in which we notice that the VM location is frequently changed at runtime. Due to the uneven distribution of users in the network, the VM shall always find the right location to minimize both the VM migration cost and the data transferring

Much more complicated control decisions can be made based on our framework. For example, besides the VM migration, the power control, channel allocation, rate control for wireless access communication and the flow balancing, and routing planning for the communication in the core networks can all be controlled based on our framework.

cost. In this case, the decision shall not only be made on current state but shall look ahead to take future possible states into consideration. The definition of the reward in our DQN-based algorithm already considers this issue and therefore can compromise between the VM migration cost and the data transfer cost. Sometimes even the immediate cost in a time slot is larger than the competitors', so choosing the right location saves VM migration cost in the future. As a result, in the long run, our framework can finally outperform the competitors.

Challenges

Although evidence shows the extraordinary ability of RL/DRL algorithms and the clear trend of applying them to pursue intelligent edge computing, there are still several challenges that need to solved.

Explainability: DRL-based solutions highly rely on the neural network, which works as a black box. First of all, it is nontrivial to design the neural network structure, even in the well-studied computer graphics area, let alone the infant edge computing resource management. Second, during neural network initialization, several hyper-parameters need to be carefully chosen for faster convergence and higher efficiency. However, how to tune the hyper-parameters is still a open issue that needs to be addressed. Therefore, the first challenge of applying RL/DRL-based solutions to edge computing resource management problems is to know how to design the neural networks and set the hyper-parameter according to a specific management problem.

Scalability: Another serious problem is known as the "curse of dimensionality," arising with the increase of environment elements to be considered in the state. As we know, the RL-based controller in our framework needs to profile the edge environment to make control decisions. It is always desirable to detail all the aspects in the environment by considering more elements. However, with the increase of the number of environment elements, the number of possible states exponentially increases. This complicates environment profiling, not only in the design of the neural network structure, but also in the agent training to experience enough number of states. Therefore, how to apply RL-based solutions to different problems and cope with ever growing network size is also a urgent problem to be solved.

Complexity: Together with the increase of network scale and state space, the correlation between decisions is becoming ever more complicated. For example, the workload balancing decision consists of both discrete and continuous actions to select the edge servers and determine their workload allocation, respectively. Moreover, the two actions are dependent on each other. When and only when a server is selected can certain workloads be allocated onto it. The problem is exaggerated when there are lots of actions in one control decision. How to tease out the relationship between these actions and appropriately design the action space is also a critical issue for the application of RL/DRL algorithms.

Flexibility: The data-driven manner makes the RL-based controller model-free, but meanwhile constrains the RL-based controller to the environment where it was trained. Unless the two environments are highly similar (e.g., similar network topology, user mobility pattern, user access mode), the controller needs to be retrained before it is actually applied. In other words, where the agent is trained is where it will be applied. This severely limits the application scope of the RL-based controller. How to share the existing experiences and knowledge of a trained controller with an untrained one is still a challenging issue. Transfer learning and federated learning are two potential machine learning technologies that can be integrated to improve RL/DRL algorithms' flexibility.

Conclusion and Future Work

In this article, we design an RL-based management framework to orchestrate the resources at the network edge. Our framework enables model-free optimization that is able to learn itself and make the appropriate control decision at runtime. To show the feasibility of our framework, we consider the problem of service migration aiming at operation cost reduction and design a DQN-based algorithm. By applying it to two different mobility models, we notice that our DQN-based algorithm can dig out the mobility regularities and accordingly make the right decision to balance the VM migration cost and the data transferring cost so as to reduce the overall cost in the long run. The capability and efficiency of our algorithm are thus proved.

As an initial work on applying RL to manage the network edge resources, our case study only shows the feasibility and efficiency of such an approach. Actually, the power of RL, as well as DRL, is far beyond this. Much more complicated control decisions can be made based on our framework. For example, besides the VM migration, the power control, channel allocation, rate control for wireless access communication and flow balancing, and routing planning for the communication in the core networks can all be controlled based on our framework. Of course, different problems require different designs based on the principle of RL algorithms. Catering to different problem characteristics, like with discrete control or continuous control, different RL algorithms with different capabilities shall be adopted as the base algorithms.

Acknowledgment

This research was supported by the NSF of China (Grant Nos. 61772480, 61602199, 61701074, 61872310).

References

[1] M. Marjanovi, A. Antoni, and I. P. arko, "Edge Computing Architecture for Mobile Crowdsensing," *IEEE Access*, vol. 6, 2018, pp. 10,662–74.
[2] M. Chen and Y. Hao, "Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network," *IEEE JSAC*, vol. 36, no. 3, Mar. 2018, pp. 587–97.

[3] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices," *IEEE JSAC*, vol. 34, no. 12, Dec. 2016, pp. 3590–3605.
[4] M. Chen *et al.*, "Label-Less Learning for Traffic Control in an Edge Network," *IEEE Network*, vol. 32, no. 6, Nov./Dec. 2018, pp. 8–14.
[5] B.-G. Kim *et al.*, "Dynamic Pricing and Energy Consumption Scheduling with Reinforcement Learning," *IEEE Trans. Smart Grid*, vol. 7, no. 5, 2016, pp. 2187–98.
[6] A. Ortiz *et al.*, "Reinforcement Learning for Energy Harvesting Decode-and-Forward Two-Hop Communications," *IEEE Trans. Green Commun. and Networking*, vol. 1, no. 3, 2017, pp. 309–19.
[7] J. Li *et al.*, "Deep Reinforcement Learning Based Computation Offloading and Resource Allocation for MEC," *Proc. WCNC*, April 2018, pp. 1–6.
[8] K. Winstein and H. Balakrishnan, "TCP Ex Machina: Computergenerated Congestion Control," *Proc. ACM SIGCOMM*, 2013, pp. 123–34.
[9] L. Chen *et al.*, "AuTO: Scaling Deep Reinforcement Learning for Datacenter-scale Automatic Traffic Optimization," *Proc. ACM SIGCOMM*, 2018, pp. 191–205.
[10] Y. He *et al.*, "Software-Defined Networks with Mobile Edge Computing and Caching for Smart Cities: A Big Data Deep Reinforcement Learning Approach," *IEEE Commun. Mag.*, vol. 55, no. 12, Dec. 2017, pp. 31–37.
[11] M. Maleki, V. Hakami, and M. Dehghan, "A Model-Based Reinforcement Learning Algorithm for Routing in Energy Harvesting Mobile Adhoc Networks," *Wireless Personal Commun.*, vol. 95, no. 3, 2017, pp. 3119–39.
[12] S. Wang *et al.*, "Deep Reinforcement Learning for Dynamic Multichannel Access in Wireless Networks," *IEEE Trans. Cognitive Commun. and Networking*, 2018.
[13] M. Yan *et al.*, "Smart Multi-RAT Access Based on Multiagent Reinforcement Learning," *IEEE Trans. Vehic. Tech.*, vol. 67, no. 5, 2018, pp. 4539–51.
[14] S. Ranadheera, S. Maghsudi, and E. Hossain, "Mobile Edge Computation Offloading Using Game Theory and Reinforcement Learning," arXiv preprint arXiv:1711.09012, 2017.
[15] C. Xu *et al.*, "Renewable Energy-Aware Big Data Analytics in Geo-Distributed Data Centers with Reinforcement Learning," *IEEE Trans. Network Science and Engineering*, 2018.

Biographies

Deze Zeng [M'14] is currently a professor in the School of Computer Science, China University of Geosciences, Wuhan. He received his Ph.D. degree from the University of Aizu, Japan. His current research interests include network functions virtualization, software-defined networking, cloud computing, and edge computing. He serves on the Editorial Boards of Elsevier *JNCA* and *FCS*.

Lin Gu [M'16] received her Ph.D. and M.S. degrees in computer science from the University of Aizu in 2015 and 2011, respectively. She received her B.S. degree from the School of Computer Science and Technology, Huangzhou University of Science and Technology (HUST), in 2009. She is now with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in the School of Computer Science and Technology, HUST. Her current research interests include cloud computing, big data, and software-defined networking.

Shengli Pan [M'16] received his B.E. and Ph.D. degrees from the University of Electronic Science and Technology of China in 2011 and 2016, respectively. From November 2014 to November 2015, he was a visiting Ph.D. student at the Swiss Federal Institute of Technology, Lausanne. He has been an assistant professor in the School of Computer Science, China University of Geosciences since 2017. His main research interests include computer networks, IoT, edge computing, and network measurements.

Jingjing Cai is currently a Master's student in the School of Computer Science and Technology, Huazhong University of Science and Technology, China. She graduated from Northeastern University and obtained her B.S. degree in 2016. Her current interests mainly focus on resource management in edge computing.

Song Guo [M'02, SM'11] received a Ph.D. degree in computer science from the University of Ottawa, Canada. He is a full professor in the Department of Computing, Hong Kong Polytechnic University. His research interests are mainly in the areas of cloud computing, and green communication and computing. He serves as an Associate Editor of *IEEE TGCN*, *IEEE TSUSC*, and *IEEE TETC*. He is a Senior Member of ACM.