Adaptive Task Offloading in Coded Edge Computing: A Deep Reinforcement Learning Approach

Nguyen Van Tam, Nguyen Quang Hieu, Nguyen Thi Thanh Van, Nguyen Cong Luong, Dusit Niyato, *Fellow, IEEE*, and Dong In Kim, *Fellow, IEEE*

***Abstract*— In this letter, we consider a Coded Edge Computing (CEC) network in which a client encodes its computation subtasks using the Maximum Distance Separable (MDS) code before offloading them to helpers. The CEC network is heterogeneous in which the helpers are different in computing capacity, wireless communication stability, and computing price. Thus, the client needs to determine a desirable size of MDS-coded subtasks and selects proper helpers such that the computation latency is within the deadline and the incentive cost is minimal. This problem is challenging since the helpers are generally dynamic and random in the computing, communication, and computing price. We thus propose to adopt a Deep Reinforcement Learning (DRL) algorithm that allows the client to learn and find optimal decisions without any prior knowledge of network environments. The experiment results reveal that the proposed algorithm outperforms the standard Q-learning and baseline algorithms in both terms of computation latency and incentive cost.**

***Index Terms*— Maximum distance separable code, coded edge computing, deep reinforcement learning.**

I. Introduction

COOPERATIVE edge computation systems allow edge nodes, i.e., helpers, to use their available resources to collaboratively execute computation intensive tasks required by a client. However, the heterogeneous nature of helpers, e.g., in terms of computation and communication capabilities, in such a system leads to the straggler effect with which the overall computation latency is limited by the computation time of the slowest participating helpers. Recently, Maximum Distance Separable (MDS)-based coding technique [1] has been proposed as an effective solution to the problem. With the (m, k) MDS code, the client first partitions the original computing task into k subtasks. Here, k is also known as the size of MDS-coded subtasks. Then, the client encodes k subtasks into $m > k$ coded subtasks before transmitting the coded subtasks to the helpers. For this, the number of coded subtask results required to retrieve the original task output is only the same as the number of uncoded subtasks, i.e., k. In other words, the client does not need to wait for the outputs from the $m - k$ slowest helpers. This eliminates the straggler effect and significantly reduces the overall computing latency of the edge computing. Thus, the MDS code can be combined with edge computing, namely Coded Edge Computing (CDC) [2].

However, one major problem of the CDC is to determine the optimal size of MDS-coded subtasks, i.e., k, and to select m proper helpers such that the computation latency meets the deadline and the incentive cost paid to the helpers is minimized. In particular, as k is large, the original task is partitioned into more subtasks and the computing time at each helper is shorter. However, the client needs to wait for more helpers to finish the subtasks to retrieve the original task output. As k is small, the client requires the small number of helpers to retrieve the original task result, but the computing time at each helper is higher. It is generally challenging for the client to find the optimal decisions since the helpers in the CEC are random and dynamic in computing capacity, communication, and computing price. In particular, the available computing resource at each helper dynamically varies over time depending on its own running tasks, and the computing price offered by the helper depends on its own cloud market. Although there have recently been few works related to the CEC, e.g., [1]–[3], they do not either address the above challenge in the CEC. The work in [1] studies the computation latency gain with MDS code based on distributed learning algorithms, i.e., data shuffling and matrix multiplication. The works in [3] and [2] address the helper selection to minimize the total computation latency. However, the incentive cost, an important factor that motivates the helpers to contribute their resources, is not accounted. Moreover, these works do not consider the dynamic and uncertainty of the CEC environment in terms of computing resource, wireless channel, and computing price of the helpers.

Thus, in this letter, we propose to leverage the Deep Reinforcement Learning (DRL) algorithm that enables the

Manuscript received August 30, 2021; accepted September 24, 2021. Date of publication September 28, 2021; date of current version December 10, 2021. This research is supported, in part by the National Research Foundation (NRF), Singapore, funded under Energy Research Test-Bed and Industry Partnership Funding Initiative, part of the Energy Grid (EG) 2.0 programme, Alibaba Group through Alibaba Innovative Research (AIR) Program and Alibaba-NTU Singapore Joint Research Institute (JRI), the National Research Foundation, Singapore under the AI Singapore Programme (AISG) (AISG2-RP-2020-019), WASP/NTU grant M4082187 (4080) and Singapore Ministry of Education (MOE) Tier 1 (RG16/20), and the MSIT (Ministry of Science and ICT), Korea, under the ICT Creative Consilience program (IITP-2020-0-01821) supervised by the IITP. The associate editor coordinating the review of this letter and approving it for publication was A. Mellouk. *(Corresponding author: Nguyen Cong Luong.)*

Nguyen Van Tam and Nguyen Cong Luong are with the Faculty of Computer Science, Phenikaa University, Hanoi 12116, Vietnam, and also with the Phenikaa Research and Technology Institute (PRATI), A&A Phoenix Group JSC, Hanoi 11313, Vietnam (e-mail: tamnvhust@gmail.com; luong.nguyencong@phenikaa-uni.edu.vn).

Nguyen Quang Hieu and Dusit Niyato are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: quanghieu.nguyen@ntu.edu.sg; dniyato@ntu.edu.sg).

Nguyen Thi Thanh Van is with the Faculty of Electrical and Electronic Engineering, Phenikaa University, Hanoi 12116, Vietnam (e-mail: van.nguyenthithanh@phenikaa-uni.edu.vn).

Dong In Kim is with the Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, South Korea (e-mail: dikim@skku.ac.kr).

Digital Object Identifier 10.1109/LCOMM.2021.3116036

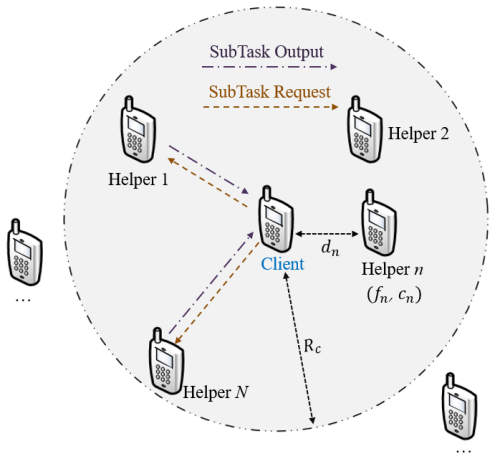

Fig. 1. An illustration of coded edge computing.

client to determine the size of MDS-coded subtasks, i.e., k, and to select m helpers to minimize the total computation latency and incentive cost without any a prior knowledge of the network environment. We first formulate the task offloading problem of the client as a stochastic problem. Then, we adopt the DRL based on Deep Q-Network (DQN) [4] to achieve the optimal policy for the client. Simulation results show that the proposed DQL method outperforms both the standard learning algorithm, i.e., the Q-learning [5], and non-learning algorithm in terms of computation latency and incentive cost.

II. System Model

We consider a CEC network as shown in Fig. 1 that consists of a client and nearby helpers. At each time slot, the client has a computing task, and we assume that there is always a set $\mathcal{N}$ that consists of N helpers available in the area of the client. In practice, N may be different in different areas that will be considered in the performance evaluation section. The area is within a circle of radius of R_c. In the time slot, helper $n \in \{1, N\}$ has an amount of available CPU resource of f_n Hz and has a computing price per time unit, denoted by c_n. Note that each helper serves not only the client but also other nearby clients. To maximize its benefit and to regulate the clients' demands, we assume that the helpers set their computing prices according to the demand-based pricing model [6]. In particular, the computing price set by helper n depends on the total computing demand in the cloud market of helper n. Here, the total computing demand is assumed to follow the Gaussian distribution [7].

At each time slot, the client has an original task of L bits which is required to be processed within a time deadline of T_L. The client partitions its own original task into k $(k \geq 1)$ subtasks with an equal size of L/k bits and then encodes them by using the (m, k) MDS code, where $k \leq m \leq N$. Such an encoding process generates m coded subtasks which have the same size, i.e., L/k bits, and enables the client to retrieve completely the original task output by collecting k different outputs of m coded subtasks. Denote $\mathcal{M}$ as a set of m selected helpers, and we have $\mathcal{M} \in \mathcal{N}$. The client sends the coded subtasks to the selected helpers. The helpers compute the tasks and transmit the outputs to the client.

The key problem of the client is to determine a desired value of k and to select m $(k \leq m \leq N)$ helpers to minimize the total computation latency and incentive cost. This is challenging since the available computing resources of the helpers are random and uncertain. The reason is that the helper has other its own tasks that can come or finish during the time that the helper executes the subtask of the client. Also, the computing price set by the helper can dynamically change according to the cloud demand of its own market.

III. Problem Formulation

To address the challenge of the client, we formulate the task offloading problem of the client as a stochastic optimization problem defined as $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} >$, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, and $\mathcal{R}$ are the state space, action space, state transition probability function, and reward function of the client, respectively.

A. State Space

The total computation latency and incentive cost depend on the size of current original task and the status of N helpers, and thus the state space of the client can be defied as follows:

$$\mathcal{S} = \{L\} \cup \prod_{n=1}^{N} \mathcal{S}_n, \tag{1}$$

where $\prod$ is the Cartesian product, and $\mathcal{S}_n$ is the state space of helper n. At the beginning of each time slot, all the N helpers send the status massages to the client. These massages are lightweight, and thus the transmission time is negligible. Then, $\mathcal{S}_n$ is defined as

$$\mathcal{S}_n = \big\{ (d_n, f_n, c_n); \times d_n \in [0, R_c], f_n \in [0, F_{\max}], c_n \in [0, C_{\max}] \big\}, \tag{2}$$

where d_n is the distance between the client and helper n, $F_{\max}$ is the maximum computing capacity of helper n and $C_{\max}$ is maximum computing price across over the helpers. Note that the distance d_n is included in the state space since it affects the transmission rate and the offloading time between the client and helper n.

B. Action Space

At each time slot, the client decides the size of the MDS-coded subtasks, i.e., k, and selects m among N helpers to execute the subtasks. Thus, the action space of the client is defined as follows:

$$\mathcal{A} = \big\{ k, \alpha_1, \alpha_2, \ldots, \alpha_N; 1 \leq k \leq N, \alpha_n \in \{0, 1\} \big\}, \tag{3}$$

where α_n is the indicator. If $\alpha_n = 1$, helper n is selected by the client to offload its MDS-coded subtask, and if $\alpha_n = 0$, helper n is not selected at the current time slot. Here, $m = \sum_{n=1}^{N} \alpha_n, k \leq m \leq N$, is the condition that the number of selected helpers must be equal or larger than the number of subtasks.

C. Reward Function

The objective of the client is to minimize the total computation latency and the incentive cost. In particular, the total computation latency is calculated according to the computation latency of each helper n that is determined as follows:

$$D_n^{\text{total}} = D_n^{\text{offload}} + D_n^{\text{compute}} + D_n^{\text{delivery}}, \tag{4}$$

where D_n^{offload} is the time that the client requires to offload (transmit) the coded subtask to helper n, D_n^{compute} is the time that helper n computes the coded subtask, and D_n^{delivery} is the time that helper n delivers its subtask output to the client. The output size is considerably small, and thus the delivery latency is negligible. The computation latency of helper n is

$$D_n^{\text{total}} = D_n^{\text{offload}} + D_n^{\text{compute}}. \tag{5}$$

To calculate D_n^{offload}, we compute the data rate from the client to helper n that is determined as follows:

$$r_n = B \log\left(1 + \frac{P_r}{\sigma^2}\right), \tag{6}$$

where B is the bandwidth assigned to helper n, P_r and σ^2 are the power of the received signal and that of the noise at the helper, respectively. The power of receiver signal can be calculated by using the free-space path loss model as follows:

$$P_r = \frac{P_t c_0^2}{(4\pi f_b d_n)^2}, \tag{7}$$

where P_t is the transmit power of the client, c_0 is the speed of light, and f_b is the carrier frequency. Therefore, the offloading time is defined as follows:

$$D_n^{\text{offload}} = \frac{L}{r_n k}. \tag{8}$$

For the computing time, D_n^{compute} is calculated based on the available CPU resource f_n of helper n and the size of the subtask assigned to the helper as $D_n^{\text{compute}} = L/f_n k$. Apart from the subtask of the client, helper n executes its own tasks. Thus, the available CPU resource in the current time slot can be greater or less than f_n depending on the running task situations. Correspondingly, the computing time of helper n can shorter or longer than $L/f_n k$. Generally, D_n^{compute} can be assumed to follow a Gaussian distribution [8] as follow:

$$D_n^{\text{compute}} = \mathcal{N}(\frac{L}{f_n k}, \tau^2). \tag{9}$$

Then, we determine D_n^{total} according to (5), and the total computation latency of the CEC network when selecting k among m coded subtask outputs is defined as follows:

$$D^{\text{output}} = \min_k \{D_n^{\text{total}}, n \in \mathcal{M}\}, \tag{10}$$

Let T_L be the time deadline of the original task. In general, T_L can be set depending on a specific application. Here, we determine T_L as the time that the client executes the task itself. This is due to the fact that the network connections to the helpers may not be available. For this, $T_L = L/f_c$, where f_c is the available CPU resource of the client. As the client outsources the helpers to execute the task, it expects that the helpers finish the task within the time deadline, i.e., $D^{\text{output}} \leq T_L$, and that D^{output} should be minimum. Thus, we can define a so-called computation reward of the client as follows:

$$R^{\text{time}} = \begin{cases} T_L - D^{\text{output}}, & \text{if } D^{\text{output}} \leq T_L \\ -T_L, & \text{if } D^{\text{output}} > T_L. \end{cases} \tag{11}$$

It can be seen from (11) that when the total latency, i.e., D^{output}, is low, then the computation reward, i.e., R^{time}, is high. When the client requests the helpers to execute its coded subtasks, it also needs to pay costs to the helpers. For simplification, the linear cost model is used with which the cost paid to helper n is $C_n = c_n D_n^{\text{compute}}$. Then, the total incentive cost paid to m selected helpers is

$$R^{\text{cost}} = \sum_{n \in \mathcal{M}} c_n D_n^{\text{compute}}. \tag{12}$$

Note that the client pays a selected helper even if the output of this helper is not collected by the client. The reason is that the selected helper needs to use its resources to compute the coded subtask. The objective of the client is to minimize the total computation latency, i.e., maximize the computation reward, and minimize the total incentive cost. Thus, the reward can be defined as follows:

$$\mathrm{R} = \mu R^{\text{time}} - \nu R^{\text{cost}}, \tag{13}$$

where μ and ν are the weights associated with the client's objectives

IV. Deep Reinforcement Learning Algorithm

Deep Reinforcement Learning (DRL) through a trial and error process has shown its ability in learning and determining optimal decisions in dynamic, random, and uncertain network environments. Especially, when the optimization problem becomes complex with the large state space. Thus, we propose to leverage the DRL algorithm to solve the optimization problem of the client. In particular, we adopt the DRL based on DQN [4] to find the optimal decisions of the client. The DRL algorithm has well presented in the literature, e.g., [4], [9]. Thus, in this section, we briefly explain how the DRL is applied to the client's problem.

Algorithm 1 Deep Q-Learning With Experience Replay [4]

Initialize θ and relay memory $\mathcal{C}$ to capacity D
for $episode = 1$ to N **do**
  for $iteration = 1$ to T **do**
    Select action a according to the ϵ-greedy policy
    Execute action a and observe reward r and next state s'
    Store experience $e =< s, a, r, s' >$ in $\mathcal{C}$
    Select N_b experiences $e_i =< s_i, a_i, r_i, s_i\prime >$ from $\mathcal{C}$ randomly
    Perform a gradient decent step on $(y_i - Q(s_i, a_i, \theta))^2$ according to (14)
  end for
end for

TABLE I
SIMULATION PARAMETERS

Parameter	Value	Parameter	Value
N	3	R_c	100 m
$F_{\max}$	6×10^8 Hz	$C_{\max}$	5×10^{-3} \$
f_c	6×10^5 Hz	B	4×10^{13} Hz
P_t	0.25 W	c_0	3×10^8 m/s
f_b	10^8 Hz	ϵ-greedy	$1.0 \rightarrow 0.1$
L	3×10^5 Bytes	μ	1.0
v	1.0	NN size	$64 \times 64 \times 64$
Optimizer	Adam	Learning rate	0.001

The DRL uses one neural network in which the input is defined in equation (1) and the output consists of the Q-values of actions. Then, the algorithm selects the action that has the highest Q-value. The DRL algorithm is implemented in two stages: the experience stage and training stage. In the experience phase, the client performs an action $a \in \mathcal{A}$ and receives an experience e. Here, action a is selected according to the ϵ-greedy policy to balance between the exploration and the exploitation. The client receives an experience that is a tuple of $e =< s, a, r, s' >$, where s' is the next state. The training stage is to train the neural network to minimize the objective function $L(\theta) = (y - Q(s, a, \theta))^2$, where y is the target value

$$y = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a', \theta). \tag{14}$$

To alleviate the data correlation issue, the DRL algorithm is combined with an experience replay memory. Algorithm 1 shows how to adopt the DRL algorithm to solve the task offloading problem of the client.

V. PERFORMANCE EVALUATION

In this section, we provide experiments to evaluate the effectiveness of our proposed DRL algorithm. For the comparison purpose, we introduce the standard Q-learning (QL) algorithm [5] and the random (RM) algorithm as baseline schemes. Note that to enable the QL algorithm to be run in the proposed network environment, three state variables, i.e., f_n, c_n, d_n, are discretized as follows. If the value of the variable is greater than its half maximum value, the value of the variable is set to 1. Otherwise, it is set to 0. With the RM algorithm, the client randomly selects k with $0 < k \leq N$ and m with $k \leq m \leq N$ to implement the $(m; k)$ MDS code. Other simulation parameters are listed in Table I. We assume that the two objectives have the same priority, and both μ and ν are set to 1. The DRL algorithm uses a deep neural network including one input, three fully-connected layers, and one output. The size of each hidden layer is 64. The sizes of the input layer and output layer are $3 \times N + 1$ and $N \times 2^N$, respectively, where N is the number of helpers. When N is set to 3, the input size is 10 and the output size is 24.

First, we discuss the total rewards obtained by the algorithms. As shown in Fig. 2(a), the proposed DRL and baseline algorithms are able to converge to stable reward values. Moreover, when the proposed DRL algorithm converges, i.e., after 2000 episodes, its total reward is higher than those obtained by the baseline algorithms. Not that with the random algorithm, the client randomly selects the helpers without learning the past experiments. Meanwhile, the computing environments are dynamic in which the computation capacity of the helpers varies from episode to episode. At the 1900-th episode, some helpers in the system may have very low computation capacities, and the client chooses these helpers for the task computation. Consequently, the reward obtained by the random algorithm at the 1900-th episode decreases suddenly.

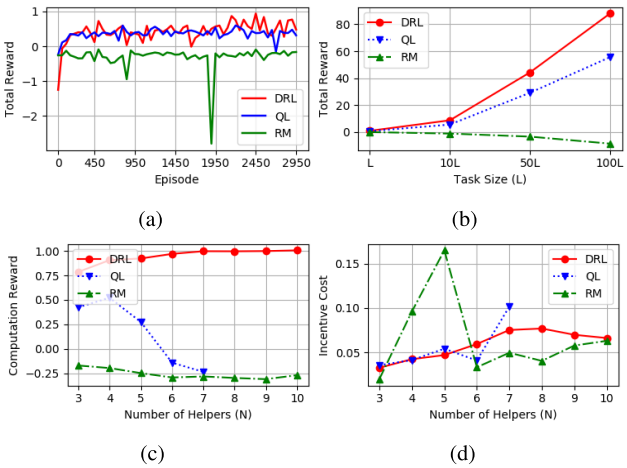

Fig. 2. (a) Total reward versus episode, (b) total reward versus task size, (c) computation reward versus the number of helpers, and (d) computation cost versus number of helpers.

The above results are for the case $L = 3 \times 10^5$ bytes and $N = 3$ helpers. In fact, the size of task and the number of helpers can be larger. Thus, we next discuss how the task size L impacts the performance, i.e., in terms of reward, of the algorithms. As shown in Fig. 2(b), as L increases, the total rewards obtained by the three algorithms increase. The reason can be explained using equation (11). Accordingly, the total reward is the difference between the total computation latency D^{output} and the time deadline T_L. When L increases, both D^{output} and T_L increase. However, due to the CPU resource constraint of the client, T_L increases faster than D^{output}, that results in the increase of the total reward. This is true in practice that the client should rent the helpers to execute the task when L increases. Figure 2(b) further shows that as L increases, the total reward obtained by the proposed DRL algorithm is much higher than those obtained by the baseline algorithms, that demonstrates the learning efficiency of the proposed DRL algorithm.

To show the scalability of the algorithms, we vary the number of helpers N in the network. Figures 2(c) and (d) illustrate the computation reward and incentive cost obtained by the algorithms as N varies. As shown in Fig. 2(c), the computation reward obtained by the proposed DRL algorithm increases as N increases. The reason is that more helpers in the network enable the client to select helpers with higher computing resources. Moreover, as observed from Fig. 2(c), the computation reward obtained by the proposed DRL algorithm is much higher than those obtained by the QL and RM algorithms. In particular, the QL algorithm does not work well as $N > 4$ helpers and cannot work as $N > 7$ due to the large state space. Note that although weights μ and ν are set to 1,

the value of the computation reward is significantly larger than the incentive cost. Thus, the DRL algorithm attempts to maximize the computation reward rather than minimizing the incentive cost. As a result, the incentive cost can fluctuates according to the dynamics of the prices offered by the helpers as shown in Fig. 2(d). Note that after training, given our computing environment, the execution time obtained by the DRL algorithm for the action decision is around 3×10^{-4}.

VI. Conclusion

In this letter, we proposed a DRL algorithm for the task offloading problem in the CEC with MDS code. First, we formulated the task offloading problem of the client as a stochastic process which determines the size of MDS-coded subtasks and selects the helpers to minimize the total computation latency and incentive cost. Under the dynamics and uncertainty of the CEC network, we proposed the DRL algorithm to solve the client's problem. The simulation results show that the proposed DRL algorithm outperforms the baseline algorithms in both computation latency and incentive cost. A general CEC system with multiple clients can be considered in the future work. In this case, the multi-agent DRL algorithm can be used.

References

[1] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[2] S. Zhao, "A node-selection-based sub-task assignment method for coded edge computing," *IEEE Commun. Lett.*, vol. 23, no. 5, pp. 797–801, May 2019.

[3] H. Park, K. Lee, J.-Y. Sohn, C. Suh, and J. Moon, "Hierarchical coding for distributed computing," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2018, pp. 1630–1634.

[4] H. Li, T. Wei, A. Ren, Q. Zhu, and Y. Wang, "Deep reinforcement learning: Framework, applications, and embedded implementations: Invited paper," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 847–854.

[5] C. J. Watkins and P. Dayan, "Technical note: Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, May 1992.

[6] D. K. S. M., N. Sadashiv, and R. S. Goudar, "Priority based resource allocation and demand based pricing model in peer-to-peer clouds," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Sep. 2014, pp. 1210–1216.

[7] S. Chen, H. L. Lee, and K. Moinzadeh, "Pricing schemes in cloud computing: Utilization-based versus reservation-based," *Prod. Oper. Manage.*, vol. 10, no. 1, pp. 1–40, Apr. 2018.

[8] W. Haynes, *Probability Distributions*. New York, NY, USA: Springer, 2013, pp. 1752–1754.

[9] T. T. Anh, N. C. Luong, D. Niyato, D. I. Kim, and L.-C. Wang, "Efficient training management for mobile crowd-machine learning: A deep reinforcement learning approach," *IEEE Wireless Commun. Lett.*, vol. 8, no. 5, pp. 1345–1348, Oct. 2019.