Deep Reinforcement Learning for Online Latency Aware Workload Offloading in Mobile Edge Computing

Zeinab Akhavan*, Mona Esmaeili*, Babak Badnava†, Mohammad Yousefi*, Xiang Sun*, Michael Devetsikiotis*, and Payman Zarkesh-Ha*
*University of New Mexico, Albuquerque, NM 87131, USA.
†University of Kansas, Lawrence, KS 66045, USA.

***Abstract*—Owing to the resource-constrained feature of Internet of Things (IoT) devices, offloading tasks from IoT devices to the nearby mobile edge computing (MEC) servers can not only save the energy of IoT devices but also reduce the response time of executing the tasks. However, offloading a task to the nearest MEC server may not be the optimal solution due to the limited computing resources of the MEC server. Thus, jointly optimizing the offloading decision and resource management is critical, but yet to be explored. Here, offloading decision refers to where to offload a task and resource management implies how much computing resource in an MEC server is allocated to a task. By considering the waiting time of a task in the communication and computing queues (which are ignored by most of the existing works) as well as tasks priorities, we propose the Deep reinforcement lEarning based offloading deCision and rEsource manageméNT (DECENT) algorithm, which leverages the advantage actor critic method to optimize the offloading decision and computing resource allocation for each arriving task in real-time such that the cumulative weighted response time can be minimized. The performance of DECENT is demonstrated via different experiments.**

***Index Terms*—Internet of Things, edge computing, resource allocation, machine learning, reinforcement learning**

I. Introduction

The growing number of Internet of Things (IoT) devices, such as smart phones and smart watches, generate huge amount of data and tasks. Normally, some of these IoT devices are resource-constrained and do not have the capacity to process the tasks locally. The mobile cloud computing (MCC) technology has been proposed to allow these devices to offload their tasks to a remote data center. However, transmitting the tasks from IoT devices to a remote data center via the Internet is expensive, leading to high and uncontrollable latency [1], thus unable to meet many IoT applications' requirements. For example, augmented reality requires the network delay to be less than 20 ms, which cannot be satisfied by MCC [2].

To reduce the network latency, mobile edge computing (MEC) has been proposed to deploy many MEC servers at the network edge. Hence, instead of offloading tasks to a remote data center, IoT devices can offload their tasks to the nearby MEC servers offering low network delay, thus potentially reducing the response time. However, computing resources of MEC servers are limited, thus offloading a task to the nearest MEC server may not always be optimal because it may incur high computing latency of executing the task, although the network delay to offload the task is minimized. Many studies have designed methods to determine whether to offload tasks from the IoT devices under a dynamic environment [3], [4]. This paper is built based on these methods by assuming the tasks have already been determined to be offloaded, but we are trying to solve the offloading decision problem, i.e., which MEC server should be selected to execute each of these task in a dynamic environment. Note that offloading decision and resource management are coupled together, meaning that whether an MEC server is suitable to execute a task depends on how much computing resource in the MEC server is allocated to the task, which is determined by the amount of remaining computing resource of the MEC server and the priority of the task. That is, if the task has low priority, i.e., low latency requirement, it is not necessary to assign all the remaining computing resource of the MEC server to the task.

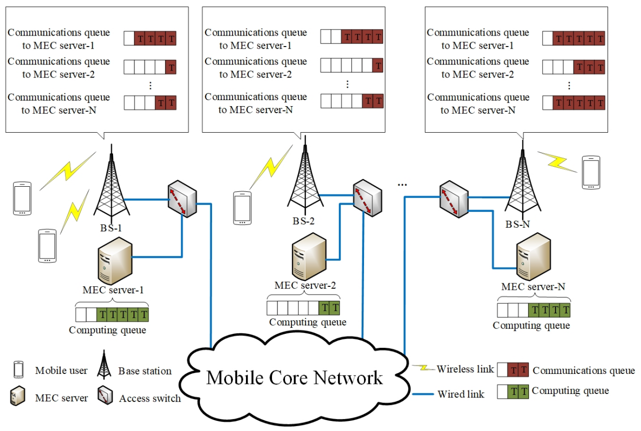

Fig. 1: The MEC architecture.

To solve the joint offloading decision and resource management problem, machine learning and non-machine learning based solutions have been developed. Non-machine learning based solutions suggest a centralized controller to solve the optimization problem and determine the offloading and re-

This work was supported by the National Science Foundation under Award CNS-2148178.

source allocation of the incoming tasks at the BS [5]–[7]. However, these solutions have the following drawbacks: 1) they only minimize the latency of the current IoT tasks by optimizing the offloading decision and resource allocation and do not consider the performance of the future IoT tasks, which may lead to the insufficient computing resources for the future IoT tasks at an MEC server, thus increasing their response time, and 2) they cannot make real-time decisions, i.e., the offloading decision and resource allocation cannot be made upon the arrival of a task. The existing machine learning based solutions employ deep reinforcement learning (DRL) to minimize the expected cumulative response time of all the tasks, which can resolve the second drawback of the non-machine learning based solutions [8], [9]. However, these solutions simplify the system by ignoring the waiting time of the tasks in the queues as well as the priorities of the tasks. Specifically, Fig. 1 shows the architecture of MEC, where each base station (BS) is attached to an MEC server via an access switch and maintains a number of communication queues, each of which buffers the arriving IoT tasks. Each MEC server executes the offloaded tasks and has a computing queue holding the tasks that are waiting for the computing resources to be released by the running tasks. The waiting time of an IoT task in the communication and computing queues would significantly affect the offloading decision and resource allocation, but yet to be considered in the existing solutions.

In this paper, we apply the advantage actor critic (A2C) method to solve the mentioned problem. Each BS observes the states of the system and determines the actions including the destination MEC server ID and the amount of the computing resources allocated to a task upon its arrival at the BS. The major contributions of the paper are listed as follows:

- We formulate the joint offloading decision and resource management problem by considering different priorities of tasks and the waiting time of the tasks in the communication and computing queues. We model this optimization problem as Markov Decision Process (MDP).
- We propose the $\underline{\text{D}}$eep reinforcement l$\underline{\text{E}}$arning based offloading de$\underline{\text{C}}$ision and r$\underline{\text{E}}$source manageme$\underline{\text{NT}}$ (DECENT) algorithm to solve the problem based on A2C.
- We demonstrate that DECENT outperforms the other two baseline approaches via extensive simulations.

The rest of this paper is organized as follows. Section II presents related work. Section III illustrates the related system model and presents the problem formulation of joint offloading decision and resource management. Section IV provides the detail of the DECENT algorithm. Section V discusses the simulation results, and Section VI concludes the work.

II. Related Work

Many works focus on the strategy to determine a task should be offloaded to the nearby MEC server or executed locally such that the response time or the energy consumption of executing the task can be minimized [10]–[12]. For example, Elgazzar *et al.* [13] proposed a decision model to evaluate whether offloading a task to the nearby MEC server improves its performance or not. The system operates by selecting a suitable resource provider to perform a task based on contextual information. Sun and Ansari [5] proposed a solution to place private virtual machines (VMs) with fixed computing resources for different mobile users to optimize the tradeoff between the migration gain and the migration cost. Assigning static computing resources to different VMs may lead to low resource utilization and increase the response time. By classifying the tasks into different IoT applications, Fan *et al.* [6] converted the task offloading problem into the application VM allocation problem. They proposed a method to dynamically adjust computing resources of different applications in each MEC server based on their workloads, thus reducing the computing delay of all tasks in the MEC server. Sun and Ansari [7] proposed a Latency aware workload offloading algorithm to optimize the offloading decision such that the average response time of the tasks can be minimized. These two papers require a centralized server to obtain the tasks from different BSs and solve an optimization problem. Also, the waiting time of the tasks in the communications queue and the priorities of the tasks are not considered in these papers. Badnava *et al.* [12] employed a Deep Q-Network to choose the best communication channel for task offloading to maximize the lifetime of a swarm of unmanned aerial vehicles (UAVs). Jia *et al.* [14] aimed to balance the workload among different geo-distributed MEC servers such that the computing latency could be minimized. Yet, the network delay of transmitting the workload among the MEC servers is ignored. Alfakih *et al.* [8] applied the reinforcement learning approach to optimize the offloading decision and bandwidth allocation to minimize the system cost, which comprises energy consumption of a mobile device and computing delay of a task. However, the paper does not dynamically allocate computing resources to different tasks. Huang *et al.* [9] proposed performance-aware resource allocation to efficiently assign computing and communication resources to users. The objective of the work is to maximize the long term performance of the system by using deep deterministic policy gradient (DDPG) to achieve the best resource allocation.

III. System Model

Fig. 1 shows the MEC architecture, where each BS is attached to an MEC server and communicates with the IoT devices in the coverage area of the BS. Each BS maintains a number of communications queues, each of which holds the tasks waiting to be transmitted to the desired MEC server, and each MEC server maintains a computing queue that holds the tasks waiting for sufficient computing resources to be released on the MEC server. Each BS determines where to offload a task and how much computing resources is allocated to the task upon its arrival such that the average response time of a task is minimized. Let $\mathcal{I}$ and $\mathcal{K}$ be the sets of the tasks and MEC servers in the system, respectively. Let i and k be the indices of the tasks and MEC servers, respectively. Note that each BS is attached to an MEC server and we will use the same index to represent a BS and its attached MEC server. In general, the response time of offloading task i to MEC sever k, denoted as T_{ik}, comprises the network delay T_{ik}^{net} and computing delay T_{ik}^{comp}, i.e., $T_{ik} = T_{ik}^{net} + T_{ik}^{comp}$. The network delay T_{ik}^{net} is the elapsed time from the arrival of task i at the BS until its delivery to an MEC server k. Furthermore, the computing delay T_{ik}^{comp} is the length of time from the arrival of task i at MEC server k to its completion. Without loss of generality, we do not consider the delay of sending the result of a task back to the IoT device.

Network Delay: The network delay of offloading task i to MEC server k comprises: 1) the transmission time T_{ik}^{trans} of task i from the BS to MEC server k via the network, i.e., $T_{ik}^{trans} = \frac{l_i}{w_k}$, where l_i is the size of task i in bits and w_k is the capacity of the path from the BS to MEC server k in bps. 2) the E2E delay between the BS and MEC server k, denoted as T_{ik}^{e2e}, and 3) the waiting time of task i in the communication queue from the BS to MEC server k, denoted as $T_{ik}^{wait\_net}$. Here, T_{ik}^{e2e} can be measured and monitored by the network controller. For instance, software defined networking (SDN) can be applied to the mobile core network and thus the SDN controller can periodically monitor and record the E2E delay between any two endpoints [15], [16]. Also, $T_{ik}^{wait\_net}$ is the time duration between the arrival of task i at the BS and its transmission starting time to MEC server k, which approximately equals the sum of the transmission time of all the tasks in the communications queue, i.e.,

$$T_{ik}^{wait\_net} = \sum_{i' \in \mathcal{I}'_{ik}} \frac{l_{i'}}{w_k}, \quad (1)$$

where $\mathcal{I}'_{ik}$ is the set of tasks in the communications queue for MEC server k when task i arrives at the BS. Hence, the network delay of offloading task i to MEC server k is

$$T_{ik}^{net} = T_{ik}^{trans} + T_{ik}^{e2e} + T_{ik}^{wait\_net} = \frac{l_i}{w_k} + T_{ik}^{e2e} + \sum_{i' \in \mathcal{I}'_{ik}} \frac{l_{i'}}{w_k}. \quad (2)$$

Computing Delay: The computing delay of offloading task i to MEC server k comprises 1) the waiting time $T_{ik}^{wait\_comp}$ of task i to be executed in the computing queue of MEC server k, and 2) the execution time $T_{ik}^{exe\_comp}$ of task i at MEC server k. Normally, $T_{ik}^{exe\_comp}$ depends on the complexity of task i (i.e., how many CPU cycles are required) and the amount of computing resources allocated to task i in MEC server k, denoted as r_{ik}. The complexity of task i is usually proportional to the size of task i [17], and thus we have $T_{ik}^{exe\_comp} = \frac{\mu l_i}{r_{ik}}$, where μ is the coefficient to map the size of a task in bits to the complexity of the task in CPU cycles. In addition, the waiting time $T_{ik}^{wait\_comp}$ depends on the number of tasks and their complexities in the computing queue when task i arrives at MEC server k, i.e., $T_{ik}^{wait\_comp} = \sum_{i'' \in \mathcal{I}''_{ik}} \frac{\mu l_{i''}}{r_{i''k}}$, where $\mathcal{I}''_{ik}$ is the set of tasks in the computing queue of MEC server k at the arrival of task i. Hence, the computing delay of task i to MEC server k is

$$T_{ik}^{comp} = T_{ik}^{exe\_comp} + T_{ik}^{wait\_comp} = \frac{\mu l_i}{r_{ik}} + \sum_{i'' \in \mathcal{I}''_{ik}} \frac{\mu l_{i''}}{r_{i''k}}. \quad (3)$$

Problem Formulation: We formulate the joint offloading decision and resource management problem as follows.

$$\textbf{P0:} \quad \underset{r_{ik}, x_{ik}}{\arg\min} \sum_{i \in \mathcal{I}} \eta_i \left(\sum_{k \in \mathcal{K}} x_{ik} \left(T_{ik}^{comp} + T_{ik}^{net} \right) \right), \quad (4)$$

$$s.t. \ \forall i \in \mathcal{I}, \sum_{k \in \mathcal{K}} x_{ik} = 1, \quad (5)$$

$$\forall i \in \mathcal{I}, \forall k \in \mathcal{K}, x_{ik} \in \{0, 1\}, \quad (6)$$

$$\forall i \in \mathcal{I}, \forall k \in \mathcal{K}, r_{ik} = \{0, R_1, R_2, R_3, \cdots R^{\max}\}, \quad (7)$$

where x_{ik} is a binary variable indicating whether task i is offloaded to MEC server k ($x_{ik} = 1$) or not ($x_{ik} = 0$) and η_i is the priority or weight of task i. A larger η_i indicates the system prefers to reduce the response time of the task and vice versa. The objective of **P0** is to minimize the overall weighted response time of all the tasks, where $\sum_{k \in \mathcal{K}} x_{ik} \left(T_{ik}^{comp} + T_{ik}^{net} \right)$ is the response time of task i. Constraint (5) indicates any task can only be offloaded to a specific MEC server. Constraint (6) means that x_{ik} is a binary variable. Constraint (7) defines the feasible values of r_{ik}, where $\{0, R_1, R_2, \cdots\}$ are the different computing resource blocks (e.g., the number of CPU cores) that can be allocated to task i in MEC server k and $R^{\max}$ is the maximum capacity of an MEC server.

P0 is nontrivial to be solved because 1) different tasks arrive at different time slots, and thus the BS cannot make the immediate decision for a task to minimize the delay if it is not aware of the future incoming tasks' information, and 2) **P0** is an NP hard problem even if the BS can predict the information of all the tasks (i.e., their arrival time, complexities, and weights). Hence, we propose to apply a DRL method to find the sub-optimal solution of **P0** in real-time.

IV. Deep Reinforcement Learning Based Offloading Decision and Resource Management

We apply MDP $(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{F}}, \boldsymbol{\mathcal{R}})$ to reformulate **P0**: 1) $\boldsymbol{\mathcal{S}}$ indicates the state space. A state at the arrival of task i, denoted as $\boldsymbol{s}_i \in \boldsymbol{\mathcal{S}}$, includes

- Weight and data size of a new task, i.e., η_i and l_i.
- Remaining computing resource of the MEC servers once new task i arrives, i.e., $\boldsymbol{C} = \{c_k \,|\forall k \in \boldsymbol{\mathcal{K}}\}$, where c_k is the remaining computing resource of MEC server k. Here, $c_k = c^{\max} - \sum_{j \in \boldsymbol{\mathcal{I}}_{ik}^{comp}} r_{jk}$, where $\boldsymbol{\mathcal{I}}_{ik}^{comp}$ is the set of tasks executed by MEC server k once task i arrives.
- Computing workload of MEC servers' computing queues once new task i arrives, i.e., $\boldsymbol{D} = \{d_k \,|\forall k \in \boldsymbol{\mathcal{K}}\}$, where d_k is the computing workload of MEC server k's computing queue, i.e., $d_k = \sum_{j \in \boldsymbol{\mathcal{I}}_{ik}^{comp\_queue}} z_j$. $\boldsymbol{\mathcal{I}}_{ik}^{comp\_queue}$ is the set of tasks in MEC server k's computing queue once new task i arrives.
- Waiting time of the communication queues once task i arrives, i.e., $\boldsymbol{B} = \{b_k \,|\forall k \in \boldsymbol{\mathcal{K}}\}$, where b_k is the waiting time of the communication queue for MEC server k, i.e., $b_k = \sum_{j \in \boldsymbol{\mathcal{I}}_{ik}^{comm\_queue}} \frac{l_j}{w_k}$. $\boldsymbol{\mathcal{I}}_{ik}^{comm\_queue}$ is the set of tasks in the communications queue once task i arrives.

2) $\boldsymbol{\mathcal{A}}$ is the set of actions for a BS to offload a task. The action set of task i, denoted as $\boldsymbol{a}_i \in \boldsymbol{\mathcal{A}}$, comprises $\boldsymbol{a}_i = \{r_{ik}, x_{ik}\}$; 3) $\boldsymbol{\mathcal{F}} : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}} \rightarrow \boldsymbol{\mathcal{S}}$ defines the state transition probability density function that maps the current states and actions into the next states; 4) $\boldsymbol{\mathcal{R}} : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}} \rightarrow \mathbb{R}$ is the reward function. The reward function for task i can be defined as the negative value of task i's response time, i.e., $r_i = -\eta_i \left(\sum_{k\in\mathcal{K}} x_{ik} \left(T_{ik}^{comp} + T_{ik}^{net}\right)\right)$.

We then design the DECENT algorithm, which is based on Advantage Actor Critic (A2C) [18], to solve the MDP problem. A2C is a DRL method combining policy-based and value-based reinforcement learning. In A2C, there are two neural networks, i.e., the actor and critic networks. The actor network provides the stochastic policy $\pi_\theta(\boldsymbol{a}_i|\boldsymbol{s}_i)$ to choose the actions $\boldsymbol{a}_i$ such that the expected cumulative reward, denoted as $J(\theta)$, is maximized. Here,

$$J(\theta) = \mathbb{E}\left[\sum_{i'=i}^{|\mathcal{I}|} \gamma^{i'} r_{i'}\right], \tag{8}$$

where θ is the parameter of the actor network, $\gamma \in [0, 1]$ is the discount factor, and $|\mathcal{I}|$ is the total number of the tasks. According to [19], the gradient of $J(\theta)$ is:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(\boldsymbol{a}_i|\boldsymbol{s}_i) A(\boldsymbol{s}_i, \boldsymbol{a}_i)], \tag{9}$$

where $A(\boldsymbol{s}_i, \boldsymbol{a}_i)$ is the Advantage function defined as

$$A(\boldsymbol{s}_i, \boldsymbol{a}_i) = r_i + \gamma V_\upsilon(\boldsymbol{s}_{i+1}) - V_\upsilon(\boldsymbol{s}_i). \tag{10}$$

Here, $V_\upsilon(\boldsymbol{s}_i)$ and $V_\upsilon(\boldsymbol{s}_{i+1})$ are the state-values with respect to task i and $i+1$ estimated by the critic network and υ is the parameter of the critic network. Hence, the actor's parameter θ is updated by the gradient descend, i.e.,

$$\theta := \theta - \beta_a \nabla_\theta J(\theta), \tag{11}$$

where β_a is the learning rate of the actor network.

The critic network in A2C is used to evaluate the actions taken by the actor network and provides the advantage value to the actor network to improve the policy. The objective of the critic network is to minimize the loss function $J(\upsilon)$, which is defined as the mean square error between the estimated state-value and the expected cumulative reward, i.e.,

$$J(\upsilon) = (r_i + \gamma V_\upsilon(\boldsymbol{s}_{i+1}) - V_\upsilon(\boldsymbol{s}_i))^2 \tag{12}$$

Denote $\nabla_\upsilon J(\upsilon)$ as the gradient of $J(\upsilon)$ with respect to the parameter υ. Then, υ is updated based on

$$\upsilon := \upsilon - \beta_c \nabla_\upsilon J(\upsilon), \tag{13}$$

where β_c is the learning rate of the critic network.

The structures of the actor and critic networks are as follows. The actor network comprises an input layer taking in the input state $\boldsymbol{s}_i = \{\eta_i, l_i, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{B}\}$, a hidden layer with 128 neurons and a relu activation function, and an output layer generating a probability distribution over actions with a softmax activation function. Likewise, critic network has an input layer taking in the state and action pair $\langle \boldsymbol{s}_i, \boldsymbol{a}_i \rangle$, a hidden layer with 128 neurons and a relu activation function, and an output layer generating the state value.

Algorithm 1: DECENT algorithm

```
1  Initialize discount factor $\gamma$, learning rates $\beta_a$ and $\beta_c$,
     and exploration rate $\epsilon$.
2  for each training episode do
3      for each task arrival do
4          Obtain the current state $\boldsymbol{s}_i$;
5          Input $\boldsymbol{s}_i$ to the actor network to calculate the
             actions $\boldsymbol{a}_i = [x_{ik}, r_{ik}]$ using $\epsilon$-greedy;
6          Calculate reward $r_i$;
7          Store transition $(s_i, a_i, r_i, s_{i+1})$ in the replay
             buffer;
8      end
9      Sample transitions from the replay buffer;
10     Update the actor neural network based on Eq. (11);
11     Update the critic neural network based on Eq. (13);
12 end
```

Algorithm 1 summarizes the DECENT algorithm, which is the process of training the actor and critic networks. Specifically, upon an arrival of task i at the BS, the actor network applies the current policy to generate the action $\boldsymbol{a}_i = \{r_{ik}, x_{ik}\}$ based on the current state $\boldsymbol{s}_i$. Note that we apply the ϵ-greedy policy where it selects the random actions with the probability of 10% and the greedy actions (that maximize the expected cumulative reward) with the probability of 90%. Based on the actions $\boldsymbol{a}_i$, we calculate the corresponding reward r_i. This process is repeated until the actor and critic networks are converged. The well-trained actor and critic networks are used to determine the actions of incoming tasks in real-time.

V. Simulation Results

In this section, we will conduct extensive simulations to validate the performance of the DECENT algorithm. Assume that there are one BS and 4 MEC servers located in different distances from the BS in km. The distances d_k from the BS to 4 MEC servers are 0, 1, 2, and 3 km, respectively, where the distance is used to calculate the E2E delay T_{ik}^{e2e} between the BS and an MEC server, i.e., $T_{ik}^{e2e} = \alpha \times d_k + \zeta$. Here, α and ζ are the coefficients, which are initially set to be 0.03 s/km and 0.03 s, respectively. The computing capacity of all the MEC servers are the same, i.e., $R^{max} = 200$ Mcycles/s. The capacity of the links from the BS to the MEC servers are the same, i.e., $w_k = 2 \times 10^9$ bits/s. The arrival of tasks follows a Poisson distribution with the average arrival rate equaling to 50 tasks. The data size of an arrival task l_i is randomly generated from a normal distribution, i.e., $l_i \sim N(3 \times 10^7\ bits, 3 \times 10^5)$. The computation intensity of a task $\mu = 0.15$ CPU cycles/bit. In addition, the number of computing resource blocks that can be assigned to an incoming task $r_{ik} \in \{10, 20, 40, 60, 80, 100, 120, 140, 160, 200\}$ Mcycles/s, and the weight of an arrival task η_i is uniformly selected, i.e., $\eta_i \sim U\{10, 20, 50, 100\}$, where a higher weight

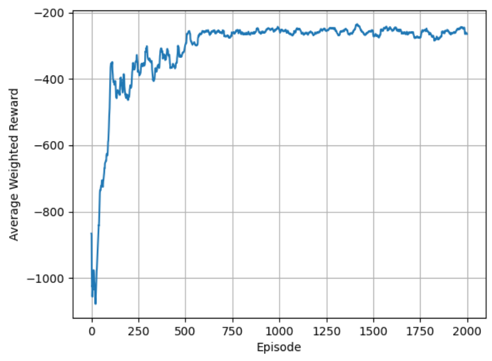

Fig. 2: Learning curve of DECENT.

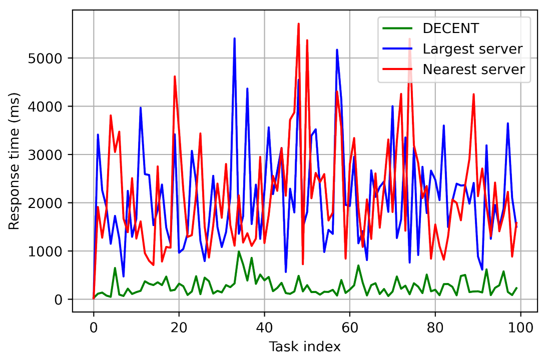

Fig. 3: Performance comparison.

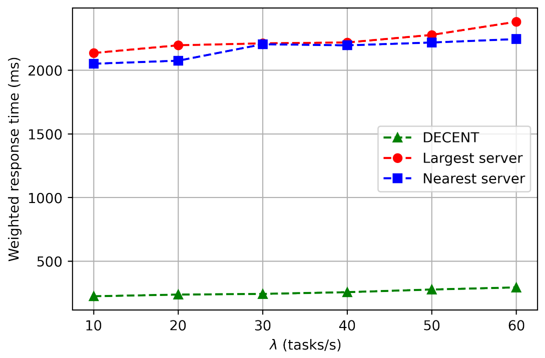

Fig. 4: Average weighted response time over average task arrival rate for different algorithms.

of a task implies the task has to be executed in a lower delay, and vice versa. Table I shows other simulation parameters.

The two baseline algorithms, i.e., nearest server and largest server, are used to compare the performance with DECENT. Here, the nearest server algorithm selects the closest MEC server (i.e., the lowest E2E delay) from the BS, and the largest server algorithm picks the MEC server with the largest remaining computing resource to offload a task.

Parameter	**Value**
Learning rate of the actor network β_a	0.0001
Learning rate of the critic network β_c	0.0002
Exploration rate ϵ	0.1
Link capacity w_k	2×10^9 bits/s
MEC server capacity R^{max}	200 Mcycle/s

TABLE I: Simulation Parameters.

Fig. 2 illustrates the learning curve for the DECENT algorithm, where DECENT can train the actor and critic networks to generate a better actions to maximize the average weighted reward in terms of minimizing the average weighted response time. The learning curves can quickly be converged around 500 episodes. Fig. 3 compares the weighted response time of different tasks for different algorithms where the average task arrival rate λ is 50 tasks/s. DECENT outperforms the baseline algorithms as the weighted response time of all the tasks can be lower than 1 second. The nearest MEC server and largest capacity MEC server algorithms, on the other hand, incurs a longer computing delay and communications delay, thus leading to a longer weighted response time.

We also examine the impact of the average task arrival rate λ on the average weighted response time among 6,400 tasks. As shown in Fig. 4, as λ increases, the average weighted response time incurred by DECENT slightly increases, while still maintaining a low level, i.e., < 500 ms. The average weighted response time increment of the other two baseline algorithms over λ is similar but much larger than DECENT, which demonstrates that DECENT achieves better offloading decision and resource management in both light and heavy workload scenarios. All the tasks can be clustered into four classes, and each class contains the tasks with the same weight. Fig. 5 shows the average network and computing delay of the tasks from different classes for DECENT. We can find that the tasks with higher weight incur lower execution time than the tasks with lower weight, which demonstrates that DECENT can adjust the offloading decision and resource allocation according to the weight of incoming tasks.

Figs. 6 and 7 show the average weighted response time among 6,400 tasks by varying the computation intensity μ and E2E delay coefficient α, respectively. Note that increasing μ and α would directly increase the E2E delay T_{ik}^{e2e} and the execution time $T_{ik}^{exe\_comp}$, respectively. DECENT always incurs the lowest average weighted response time than the two baseline algorithms in different scenarios. It is interesting to see that the performance of the largest server algorithm in Fig. 7 is significantly degraded as α increases. This is because increasing α increases T_{ik}^{e2e}, which may dominate the response time, and so offloading tasks to the nearby servers is preferred to reduce the weighted response time. Yet, the largest server algorithm does not consider the network delay, thus leading to significant performance degradation.

VI. Conclusion

By considering the waiting time of a task in the communication and computing queues as well as different priorities of the tasks, this paper proposed the DECENT algorithm allowing each BS to determine the offloading decision and computing resource allocation for each arrival task in real-time such that the cumulative weighted response time is minimized. As compared to the two baseline algorithms, DECENT has

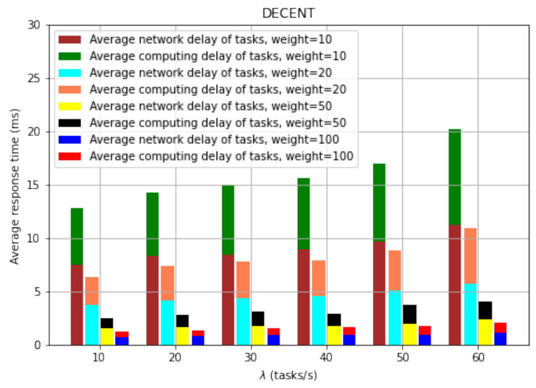

Fig. 5: Average response time w.r.t λ and weights of the tasks.

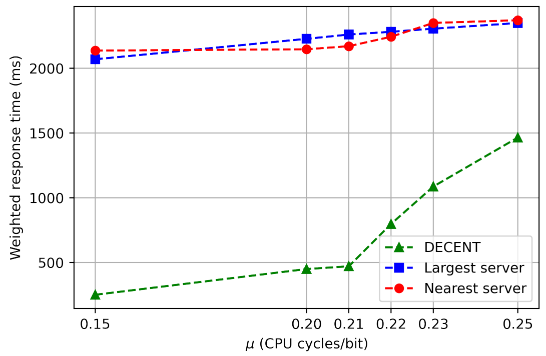

Fig. 6: Average weighted response time over μ.

been demonstrated to have lower response time in different scenarios via extensive simulations. Also, DECENT is capable of adjusting the offloading decision and computing resource allocation based on the weights of the incoming tasks to minimize the cumulative weighted response time.

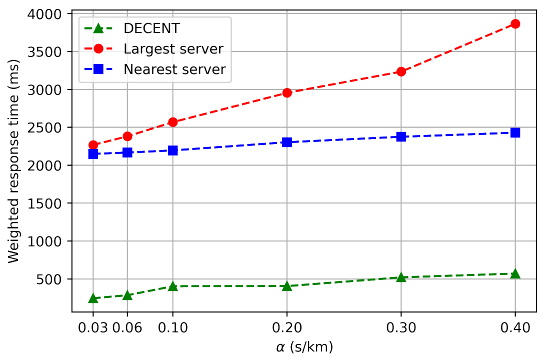

Fig. 7: Average weighted response time over α.

References

[1] X. Sun and N. Ansari, "Edgeiot: Mobile edge computing for the internet of things," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22–29, 2016.
[2] M. Abrash, "Latency–the sine qua non of ar and vr," *Blog post, Dec*, 2012.
[3] X. Sun and N. Ansari, "Adaptive avatar handoff in the cloudlet network," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 664–676, 2019.
[4] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 12, no. 9, pp. 4569–4581, 2013.
[5] X. Sun and N. Ansari, "Primal: Profit maximization avatar placement for mobile edge computing," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.
[6] Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based iot," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2146–2153, 2018.
[7] X. Sun and N. Ansari, "Latency aware workload offloading in the cloudlet network," *IEEE Commun. Lett.*, vol. 21, no. 7, pp. 1481–1484, 2017.
[8] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa," *IEEE Access*, vol. 8, pp. 54 074–54 084, 2020.
[9] B. Huang, Z. Li, Y. Xu, L. Pan, S. Wang, H. Hu, and V. Chang, "Deep reinforcement learning for performance-aware adaptive resource allocation in mobile edge computing," *Wireless Communications and Mobile Computing*, vol. 2020, 2020.
[10] Y. Liu, M. J. Lee, and Y. Zheng, "Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system," *IEEE Trans. Mobile Comput.*, vol. 15, no. 10, pp. 2398–2410, 2015.
[11] A. Mukherjee, D. De, and D. G. Roy, "A power and latency aware cloudlet selection strategy for multi-cloudlet environment," *IEEE Trans. Cloud Comput.*, vol. 7, no. 1, pp. 141–154, 2016.
[12] B. Badnava, T. Kim, K. Cheung, Z. Ali, and M. Hashemi, "Spectrum-aware mobile edge computing for uavs using reinforcement learning," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, 2021, pp. 376–380.
[13] K. Elgazzar, P. Martin, and H. S. Hassanein, "Cloud-assisted computation offloading to support mobile services," *IEEE Trans. Cloud Comput.*, vol. 4, no. 3, pp. 279–292, 2014.
[14] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet load balancing in wireless metropolitan area networks," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.
[15] C. Yu, C. Lumezanu, A. Sharma, Q. Xu, G. Jiang, and H. V. Madhyastha, "Software-defined latency monitoring in data center networks," in *International Conference on Passive and Active Network Measurement*. Springer, 2015, pp. 360–372.
[16] X. Sun and N. Ansari, "Green cloudlet network: A sustainable platform for mobile cloud computing," *IEEE Trans. Cloud Comput.*, vol. 8, no. 1, pp. 180–192, 2020.
[17] Q. Fan, J. Bai, H. Zhang, Y. Yi, and L. Liu, "Delay-aware resource allocation in fog-assisted iot networks through reinforcement learning," *IEEE Internet of Things Journal*, vol. 9, no. 7, pp. 5189–5199, 2022.
[18] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.
[19] C. Yoon, "Understanding actor critic methods and a2c," [Online]. Available: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f.
[20] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.
[21] Y. Wei, F. R. Yu, M. Song, and Z. Han, "Joint optimization of caching, computing, and radio resources for fog-enabled iot using natural actor–critic deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2061–2073, 2018.
[22] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in iot edge computing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1133–1146, 2020.
[23] J. Yao and N. Ansari, "Power control in internet of drones by deep reinforcement learning," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.
[24] X. Liu, Z. Qin, and Y. Gao, "Resource allocation for edge computing in iot networks via reinforcement learning," in *ICC 2019-2019 IEEE international conference on communications*. IEEE, 2019, pp. 1–6.
[25] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 11, pp. 5141–5152, 2019.
[26] X. Chen, J. Wu, Y. Cai, H. Zhang, and T. Chen, "Energy-efficiency oriented traffic offloading in wireless networks: A brief survey and a learning approach for heterogeneous cellular networks," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 4, pp. 627–640, 2015.