Delay-sensitive Task Scheduling with Deep Reinforcement Learning in Mobile-edge Computing Systems

Hao Meng¹,a, Daichong Chao¹,* , Qianying Guo¹,b and Xiaowei Li¹,c

¹Beijing Advanced Innovation Center for Future Internet Technology, Beijing University of Technology, Beijing, China

Email: a menghao0528@126.com; * chaodaichong@163.com; b guo_qianying@163.com; c lixiaowei_bjut@163.com

Abstract. Mobile-edge computing (MEC) is considered to be a new network architecture concept that provides cloud-computing capabilities and IT service environment for applications and services at the edge of the network, and it has the characteristics of low latency, high bandwidth and real-time access to wireless network information. In this paper, we mainly consider task scheduling and offloading problem in mobile devices, in which the computation data of tasks that are offloaded to MEC server have been determined. In order to minimize the average slowdown and average timeout period of tasks in buffer queue, we propose a deep reinforcement learning (DRL) based algorithm, which transform the optimization problem into a learning problem. We also design a new reward function to guide the algorithm to learn the offloading policy directly from the environment. Simulation results show that the proposed algorithm outperforms traditional heuristic algorithms after a period of training.

1. Introduction

With the advent of the 5G era and the development of computer application technology, a large number of new applications and services, which are computation-intensive and delay-sensitive, are emerging, e.g., augmented reality (AR), virtual reality (VR) and ultra-high-definition (UHD) video. This poses a huge challenge to the computing power and battery life of mobile devices in spite of the continuous development of processor and battery technology. The emergence of mobile edge computing (MEC) brings new ideas to the solution of the above issues. MEC sinks IT services to the mobile access networks side, thus a service environment with high performance, low latency and high bandwidth is created for users. By offloading a part of computing tasks from mobile devices to the nearby MEC servers, the average slowdown of tasks will be shortened and the user experience will be greatly improved.

The joint resource management of radio and computational plays an important role in realizing low-latency and energy efficient MEC. In single-user MEC Systems, there are three frequently-used task models: deterministic task model with binary offloading, deterministic task model with partial offloading, and stochastic task model [1]. The general guidelines for determining the binary offloading decision are proposed in [2], [3] to minimize the energy consumption and computation latency of mobile devices. In [4], with the input data transmission time as the optimization variable, the problem of minimizing the transmission energy under the constraint of calculation deadline was established, which is convex and can be solved in closed form. [5], [6] studies the partial offloading task models. In [5], the offloading ratio, transmission power and CPU-cycle frequency are jointly optimized for the
purpose of minimizing the mobile-energy consumption under latency constraints, assuming that the data of task can be arbitrarily divided for local and remote execution. The integer programming approach is used in [6] to jointly optimize scheduling and cloud offloading decisions, minimizing energy consumption from computational execution. In stochastic task model, tasks arrive randomly and are cached in the queue for scheduling, it is necessary to consider the long-term performance of the system from a global perspective. [7] designs a latency-optimal task scheduling strategy based on the theory of Markov decision process (MDP), which take the states of local-processing, transmission unit and the length of task buffer queue in to consideration, the simulation results show that it outperforms the traditional greedy policies. A Lyapunov optimization-based algorithm was proposed in [8] in order to jointly decide the task allocation, offloading policy, CPU clock speed and selected network interface selection.

In this paper, we consider a single-user MEC system with random delay-sensitive task arrivals, where the computational data of each task to be offloaded has been determined. On the basis of foregoing description, we design an algorithm based on deep reinforcement learning for the objective of optimizing the long-term performance (e.g., average slowdown and average timeout period) of MEC systems by jointly task scheduling and resource allocation. We also propose a new reward function to guide the algorithm move towards the right objective. Simulation result proves that the deep reinforcement learning algorithm can get better performance of lower average delay and average timeout period than the traditional heuristic algorithms.

The rest of this paper is organized as follows: The Section 2 introduces the background of deep reinforcement learning, the system model will be described in Section 3. The design of deep reinforcement learning based task scheduling algorithm is presented in Section 4, the simulation results is shown in Section 5, and Section 6 concludes the paper.

2. Background

Learning is the process by which an agent improves its performance. Machine learning studies how an agent implemented by a computer program can improve its processing performance by learning, and Reinforcement Learning (RL) is a kind of machine learning, which attracts great attention in the field of machine learning and artificial intelligence due to its biological relevance and learning autonomy.

In reinforcement learning, as illustrated in Figure 1, an agent learns how to optimally match states and actions with environment by trial and error in order to get the maximum cumulative discounted reward.

Generally, the reinforcement learning tasks can be described by Markov Decision Process (MDP), and the MDP can be denoted as a five-tuple: \( M = \{S, A, P, \pi, R\} \), where \( S \) indicates the state space, \( A \) is action space, \( P_a(s, s') \) describes the transition probability of the state with action \( a \), \( \gamma \) is discount factor, which varies from 0 to 1 and represents the degree of farsightedness of the agent, \( R \) is reward function.

![Figure 1. Deep Reinforcement Learning model with DNN approximator](image)

The rules for how to choose actions are called policy, a policy is a probability distribution in which a state is mapped to an action, if the agent follows the policy \( \pi \) at time \( t \), then \( \pi(s, a) \) represents the
probability of performing action $a$ under state $s$. In some simple cases, we can use tables to store the mapping between states and actions, but in practice, the magnitude of this mapping is very large, the traditional way of storing the state action pairs is not feasible, so it is necessary to use function approximators. A function approximator usually possesses a limited quantity of adjustable parameters. Hence, a policy with adjustable parameter can be denoted as $\pi_\theta(s,a)$. In this paper, we use Deep Neural Network (DNN), which has made a huge breakthrough in recent years, as a function approximator, and update the parameters with a policy gradient method. Recall that the goal of reinforcement learning is to maximize the cumulative discounted reward, $E[\sum_{t=0}^\infty \gamma^t r_t]$, the gradient of this goal given by[9]:

$$\nabla_\theta E_{\pi_\theta}[\sum_{t=0}^\infty \gamma^t r_t] = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)Q^{\pi_\theta}(s,a)]$$

(1)

where $Q^{\pi_\theta}(s,a)$ is the expected value of state action pair $(s,a)$, $\gamma$ is discount factor. In simple Monte Carlo Method [10], multiple trajectories are sampled by agent and the agent uses the calculated cumulative discounted reward as unbiased estimate of $Q^{\pi_\theta}(s,a)$, and then update the parameters via gradient descent:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s,a) v_t$$

(2)

where $\alpha$ is the learning rate, $v_t$ is the cumulative discounted reward in one of the trajectory sampled by Monte Carlo Method. In our implementation, we subtract the average value of multiple sampling for each $v_t$ to reduce the variance of the gradient. More details will be described below.

### 3. System model

The single-user MEC system contains a mobile device and a MEC server. The mobile device runs multiple computationally intensive and delay-sensitive independent tasks and the MEC server assists in performing computational tasks and provides computing power. As shown in Figure 2, the mobile device mainly includes a task buffer queue $Q$, a task scheduler, a local execution unit and a transmission unit. Among them, the task buffer queue $Q$ is used to cache the to-be-processed tasks of the mobile device; the scheduler is configured to schedule the tasks from $Q$ to enter the transmission unit and the execution unit for processing; the local execution unit can process multiple tasks concurrently until it’s utilization rate is 100%, also, the transmission unit can transmit multiple tasks at the same time until the bandwidth occupancy is full. The MEC server is deployed on the mobile access network side and communicates with the mobile device over the wireless channel. We assume that the MEC server has sufficient computing resources and the computing speed is $k$ times faster than the mobile devices, so that it can handle multiple tasks simultaneously, and we do not need to consider the waiting delay on MEC server. In addition, it is assumed that the calculation results are very small, so the feedback delay can be neglected, the transmission power of the mobile device’s transmission unit is $p_{trans}$, and the distance between the mobile device and the MEC server is $L$.

![Figure 2. The structure of single-user MEC system](image-url)
execution unit due to device-dependent reasons (e.g., the need to interact with users or access local I/O devices) and the other part of computation, which is computation intensive and device-independent, needs to be offloaded to the MEC server for execution, so we set up a resource profile $v_{res,i}$ for each task to describe the resources requirements. Hence, each task in $Q$ is characterized by a six-tuple parameters, $f_i = \{t_{\text{enter},i}, t_{\text{start},i}, t_{\text{fin},i}, v_{res,i}, t_{\text{proc},i}, t_{\text{overtime},i}\}$, where $t_{\text{enter},i}$ is the time step when task $f_i$ entering the buffer queue, $t_{\text{start},i}$ is the start execution time step of task $f_i$, $t_{\text{fin},i}$ is the completion time step of task $f_i$, $v_{res,i}$ represents the resource profile of task $f_i$ in transmission unit and local execution unit and it can be denoted as $v_{res,i} = \{v_{e,i}, v_{c,i}\}$, where $v_{e,i}$ and $v_{c,i}$ are the occupancy ratio of local execution unit and transmission unit, respectively. $t_{\text{proc},i}$ indicates the expected execution time of the task $f_i$, $t_{\text{overtime},i}$ describes the timeout threshold for the task $f_i$. Besides, denote $t_{\text{cur}}$ as the current time step of the MEC system, then the waiting time of task $f_i$ is $w_i = t_{\text{cur}} - t_{\text{enter},i}$, the timeout period can be denoted as $\delta_i = \max(0, t_{\text{cur}} + t_{\text{proc},i} - t_{\text{enter},i} - t_{\text{overtime},i})$, the slowdown of task $f_i$ can be calculated as $\frac{t_{\text{fin},i} - t_{\text{enter},i}}{t_{\text{proc},i}}$.

4. Design of algorithm

This paper adopts the deep reinforcement learning algorithm to model the system environment. The algorithm will be analysed from four aspects: state space, action space, reward, and function approximator.

4.1. State space

We denote the state of the single-user MEC system as $S$, it mainly consists of the resource profile of tasks in the queue and the resource allocation status of execution unit and transmission unit of mobile device in the current and future $T$ timesteps[11], as described in Figure 3, the horizontal axis indicates time, the rightmost column of the vertical axis shows the allocation status of CPU in local execution unit and the bandwidth of transmission unit in the future $T$ timesteps, the different colors in the allocation status image represent different tasks, which are being performed. For example, the green task in the image indicates that it will continue to occupy two unit of CPU and one unit of bandwidth in the next four timesteps, also, the orange task is scheduled to use one unit of CPU and two units of bandwidth in the next three timesteps. The first two columns of the vertical axis represent the resource profile of tasks in the task queue, for example, the task 1 requires one unit of CPU and three units of bandwidth, and it is expected to execute four timesteps. Since the deep reinforcement learning algorithm requires a fixed input dimension, but the task number of the queue $Q$ is uncertain, so we take the first $D$ task’s resource profile as the component of the state space. Intuitively, it is sufficient to limit attention to the first-going tasks, because a reasonable policy may prefer tasks that waits longer, and it also facilitates the definition of the following action space. Specially, if the number of jobs in the queue is no larger than $D$, we will construct tasks that their resource profiles are null to ensure the integrity of the input in the deep reinforcement algorithm.
4.2. Action space
As can be seen from the aforementioned system model, the scheduler needs to schedule as many tasks as possible in one timestep to minimize the average slowdown and average timeout period of tasks in the buffer queue. But if we do this in normal way, it may make the action space very large, which greatly increases the complexity of the action space. Here, we adopt a trick [11]: the time step is frozen when the task is properly scheduled (the scheduler takes a valid action) until the scheduler takes a null action or an invalid action in the current time step. At this point, the current timestep moves forward and then repeats the steps mentioned above. In this way, it not only meets the requirement of scheduling more tasks in one timestep, but also reduces the complexity of action space. In addition, we have defined the depth of action space as \( D \) in the state space, therefore, the action space can be described as:

\[
A = \{0, 1, 2, ..., i, ..., D\}
\]  

(3)

Where 0 is the null action, action i indicates that the scheduler will schedule i-th task into the system for processing.

4.3. Rewards
To minimize the average slowdown and average timeout period of tasks in the task queue while ensuring fair scheduling, we define the reward function as

\[
R = \beta \times R_s + (1 - \beta) \times R_o
\]  

(4)

where \( R_s = \sum_{i \in J} \frac{1}{t_{proc}} \) is the reward signal of slowdown, \( R_o = \sum_{i \in J} \left\{ \begin{array}{ll}
0 & o_i = 0 \\
\frac{1}{q_i} & o_i > 0
\end{array} \right. \) is the reward signal of timeout period, \( J \) indicates all tasks in the mobile device, including tasks waiting in the queue and performing in the execution unit, \( \beta \) is the tradeoff factor, which satisfies \( \beta \in [0, 1] \). If we set \( \beta = 1 \) and the discount factor \( \gamma = 1 \), then the reward function only consider the effect of slowdown, maximizing the cumulative discounted reward mimics minimizing the average slowdown. When we set \( \beta = 0 \), the reward function only take the timeout period in to consideration, the longer the timeout period of tasks in the task queue, the smaller the rewards won. When \( \beta \) satisfies \( \beta \in (0, 1) \), the objective of the reward function is minimizing timeout period while minimizing average slowdown of jobs in the queue.

4.4. Function approximator
The DNN is used as function approximator. It contains an input layer, two hidden layers and an output layer. The dimensions of the input layer and the output layer are the same as those of the state space and the action space, and the dimensions of the two hidden layers can be represented by H1 and H2, respectively. After each hidden layer, ReLU [12] is used as an activation function to increase the nonlinearity of the neural network. Meanwhile, we use Softmax [13] to output the probability distribution of the action in the current state. Therefore, the structure of function approximator can be expressed as: input->H1->Relu->H2->Relu->output->softmax.

5. Simulation results
In order to estimate the performance of the proposed algorithm, we performed a simulation experiment. For the mobile device, we discretize the CPU of execution unit and the bandwidth of transmission unit into \( K \) and \( M \) units, respectively and we set \( K=10 \) in this simulation. For task \( f_i = \{t_{enter,i}, t_{start,i}, t_{fin,i}, v_{res,i}, t_{proc,i}, t_{overtime,i}\} \), the values of \( t_{enter,i}, t_{start,i} \) and \( t_{fin,i} \) are dynamically assigned when the algorithm runs; The \( v_{res,i} = \{v_e,i, v_t,i\} \), \( t_{proc,i} \) and \( t_{overtime,i} \) settings are assumed to be uniformly distributed. Specifically, for \( v_{res,i} \), we randomly select one of the two as the dominant resource occupier and it is assumed to obey Unif([M/2, M]), another is Unif([1, M/3]); for \( t_{proc,i} \), we set the probability of 0.8 obeying Unif([1, \( t_{proc,max} / 5 \)]) and 0.2 obeying...
Unif\((2 * t_{\text{proc,max}}/3, t_{\text{proc,max}})\), where \(t_{\text{proc,max}} = 15\); for \(t_{\text{overtime.i}}\), we set it to obey Unif\((1.5 * t_{\text{proc}}, 4 * t_{\text{proc}})\). Besides, we set \(T = 20\), \(H_1 = H_2 = 32\), \(D = 5\), \(\beta = 0.5\).

Four traditional scheduling algorithms are compared: Packer [14], FCFS (First-come, First-served), Random, and SJF (Short Job First). Packer allocate jobs according to the matching degree of task’s resource requirements and the idle resources of the system. FCFS states that the algorithm schedules jobs according to their arrival order. The Random algorithm schedules tasks in a random manner. SJF means that the shortest duration of job in the current action list is allocated first.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig4.png}
\caption{The mean and max cumulative discounted reward across iteration.}
\end{figure}

We simulated the deep reinforcement learning based task scheduling and offloading algorithm through the modelling of experimental scenarios. Figure 4 shows the maximum and mean cumulative discounted rewards for all Monte Carlo runs during the iteration. As expected, both rewards are gradually increasing as the training progresses, but there is a clear gap between the two rewards at the beginning of the training, and the gap gradually decreases or even disappears later in the training. The reason for this phenomenon is that the policy of reinforcement learning is a probability distribution over all possible actions. When a sequence of actions sampled by Monte Carlo method is better than the other, this gap will occur. This also shows that there is room for further learning by DNN. When this gap narrows, as we see near the 1500\(^{th}\) iteration, the model has converged. In Figure 5, the changes in the average slowdown and average timeout periods during training are drawn, compared with other traditional algorithms, both of them gradually decrease as the training progresses. At the beginning of training, the average slowdown of our algorithm is 10.42, only better than Packer (16.80) and FCFS(14.98). When the model converges, it has dropped to 4.58, while Random is always 10.31, SJF is 8.34. Similarly, for the average timeout period, our algorithm has a larger timeout rate than some traditional algorithms when the task generation rate is lower than 0.6, however, with the increase of task generation rate, our algorithm has a smaller average timeout period than other algorithms when the task generation rate is above 0.7 (except 0.8).
6. Conclusion

In this paper, we propose a deep reinforcement learning based delay-sensitive task scheduling algorithm to solve the task scheduling problem in single-user MEC system, the goal is to speed up task processing time while reducing task overtime period. At the same time, we innovatively propose a new reward function to guide the algorithm to explore, utilize and learn from environment. The simulation results show that the proposed algorithm performs better than the traditional algorithms after a period of adaptive policy learning. In the future research, we will take more factors such as the distance between mobile device and MEC server, the energy consuming by mobile device and other related factors in to consideration.

References

[1] Mao Y, You C, Zhang J, Huang K, and Letaief K B 2017 Mobile edge computing: survey and research outlook
[2] Kumar K and Lu Y H 2010 Cloud computing for mobile users: can offloading computation save energy? Computer 43(4) pp 51-56
[3] Kumar K, Liu J, Lu Y H and Bhargava B 2013 A survey of computation offloading for mobile systems Mobile Networks and Applications 18(1) pp 129-40
[4] Barbarossa S, Sardellitti S and Di Lorenzo P 2014 Communicating while computing: distributed mobile cloud computing over 5g heterogeneous networks IEEE Signal Processing Magazine 31(6) pp 45-55
[5] Wang Y, Sheng M, Wang X, Wang L and Li J 2016 Mobile-edge computing: partial computation offloading using dynamic voltage scaling IEEE Transactions on Communications p 1-1

[6] Mahmoodi S E, Uma R N and Subbalakshmi K P 2016 Optimal joint scheduling and cloud offloading for mobile applications IEEE Transactions on Cloud Computing p 1-1

[7] Liu J, Mao Y, Zhang J and Letaief K B 2016 Delay-optimal computation task scheduling for mobile-edge computing systems

[8] Kwak J, Kim Y, Lee J and Chong S 2015 Dream: dynamic resource and task allocation for energy minimization in mobile cloud systems IEEE Journal on Selected Areas in Communications 33(12) pp 2510-23

[9] Sutton R S and Barto A G 1998 Reinforcement learning: an introduction IEEE Transactions on Neural Networks 9(5) p 1054-1054

[10] Sutton R S, McAllester D A, Singh S P and Mansour Y 1999 Policy gradient methods for reinforcement learning with function approximation Neural Information Processing Systems 12 MIT

[11] Mao H, Alizadeh M, Menache I and Kandula S 2016 Resource Management with Deep Reinforcement Learning Acm Workshop on Hot Topics in Networks

[12] Glorot X, Bordes A and Bengio Y 2010 Deep Sparse Rectifier Neural Networks Journal of Machine Learning Research p 15

[13] Priddy K L and Keller P E 2005 Artificial Neural Networks: An Introduction

[14] Grandl R, Ananthnarayanan G, Kandula S, Rao S and Akella A 2014 Multi-resource packing for cluster schedulers Acm Conference on Sigcomm