# Computational Offloading Optimisation for UAV-assisted Mobile Edge Computing

Animesh Sinha, Khushi Prasad

Indian Institute of Technology Patna

## 1 Introduction

### 1.1 Motivation

With time, computation-intensive applications like VR/AR, online gaming and telemedicine, will become more prominent. However, these mobile applications customarily demand considerable computing resources and have a high energy consumption, prerequisites which current User Equipments(UEs) fail to meet due to having limited computing resources, storage and battery capacity. This problem can be solved with the emergence of Mobile Cloud Computing (MCC) by offloading computation to the cloud via mobile networks.

Since cloud servers are generally physically distant from the UE, it may result in high transmission delay and have an adverse effect on user experience. To minimise the delay, a new computation paradigm has emerged in the field of mobile computing namely, Mobile Edge Computing (MEC). MEC can increase the proximity between the cloud computing resources and the UE, so as to effectively reduce communication delay and energy dissipation.

The MEC services provided by fixed infrastructures, which was the focus of earlier studies on MEC, cannot effectively work in the scenarios where communication facilities are meagerly allocated or when disasters take place abruptly. Recently, researchers have turned to Unmanned Aerial Vehicle (UAV) due to its high mobility and versatility. Despite detailed studies and experimentations, UAV-assisted MEC systems still present many challenges. Thus, adaptive link selection and task offloading problems are very crucial in UAV-assisted MEC systems.

## 2 System Model

We consider a UAV-assisted MEC system, which consists of a UAV mounted with a nano MEC server along with K UEs. The system operates in discrete time with equally divided time slots. The UAV provides communication and computing resources to all UEs, but only to one at a time. Limited computing

capability compels the UE to offload a portion of the computing tasks to the MEC server at the UAV via wireless network.

## 2.1 Communication Model

The communication period T is divided into I time slots, during which the UAV provides computing services to one UE at a time. We assume that the UEs move randomly in the area at a low speed. In each time slot, the UAV hovers in a fixed position, at a fixed altitude H, and then communication is established with one of the UEs. After offloading a portion of its computing tasks to the server, the remaining tasks are executed locally by the UE.

Initially, the coordinate of the UAV is given by $q(i) = [x(i), y(i)]^T \in \mathbf{R}^{2\times 1}$ and the end coordinate is $q(i+1) = [x(i+1), y(i+1)]^T \in \mathbf{R}^{2\times 1}$ at time slot i $\in \{1, 2, ..., I\}$. The expression for the channel gain of the line-of sight link between the UE k and the UAV is

$$g_k(i) = \alpha_0 d^{-2}(i) = \frac{\alpha_0}{||\mathbf{q}(i+1) - \mathbf{p}(i+1)||^2 + H^2}. \tag{1}$$

where $\alpha_0$ denotes the channel gain at a reference distance of 1m, and $d_k(i)$ is the Euclidean distance between the UAV and UE. Due to the presence of obstacles, the wireless transmission rate is shown by:

$$r_k(i) = Blog_2(1 + \frac{P_{up} g_k(i)}{\sigma^2 + f_k(i) P_{NLOS}}), \tag{2}$$

where B denotes the communication bandwidth, $P_{up}$ denotes the transmit power of the UE in the upload link, $\sigma^2$ denotes the noise power, $f_k(i)$ is the indicator of whether there is blockage between the UAV and UE at time slot i and $P_{NLOS}$ denotes the transmission loss.

## 2.2 Computation model

Partial offloading of tasks takes place in our MEC system. Let $R_k(i) \in [0, 1]$ be the ratio of tasks offloaded to the server, while the remaining tasks are executed locally by the UE. Therefore, local execution delay of a UE can be given by:

$$t_{local,k}(i) = (1 - \frac{R_k(i)) D_k(i) s}{f_{UE}}). \tag{3}$$

where $D_k(i)$ is the computing task size of UE k at the $i^{th}$ time slot, s represents the number of CPU cycles required, and $f_{UE}$ is the UE's computing capability.

The processing delay and the power consumption of MEC server can be divided into two parts. The first is transmission delay, which consists only of the uplink delay since the size of the computation result provided by the server is negligible:

$$t_{tr,k}(i) = \frac{R_k(i) D_k(i)}{r_k(i)}. \tag{4}$$

The second is the delay caused by the computation at MEC server:

$$t_{UAV,k}(i) = \frac{R_k(i)D_k(i)s}{f_{UAV}} \tag{5}$$

where $f_{UAV}$ is the computation frequency of the server CPU.

### 2.3 Problem formulation

Thus, we formulate the optimisation problem in our UAV-assisted MEC system. Our aim is to minimise the maximum processing delay of all the UEs in order to ensure the efficient utilisation of our limited computation resources. This can be done by jointly optimising user scheduling, UAV mobility and resource allocation in our system. Therefore, our optimisation problem can be represented as:

$$min_{\alpha_k(i),q(i+1),R_k(i)} \sum_{i=1}^{l} \sum_{k=1}^{K} \alpha_k(i) max\{t_{local,k}(i), t_{UAV,k}(i) + t_{rr,k}(i)\} \tag{6}$$

## 3 DDPG based computation offloading optimisation

DQN algorithm can solve the problem of high-dimensional state spaces, however it can not handle continuous action spaces. Therefore, the DDPG algorithm has an advantage in this situation. As a model-free off-policy actor-critic algorithm using DNN, DDPG algorithm can learn polices in continuous action spaces. The actor-critic algorithm consists of a policy function and a Q-value function. The policy function acts an actor and generates actions. These actions are then evaluated by the Q-value function, which acts as a critic to the actor's performance and directs the actor's follow-up actions.

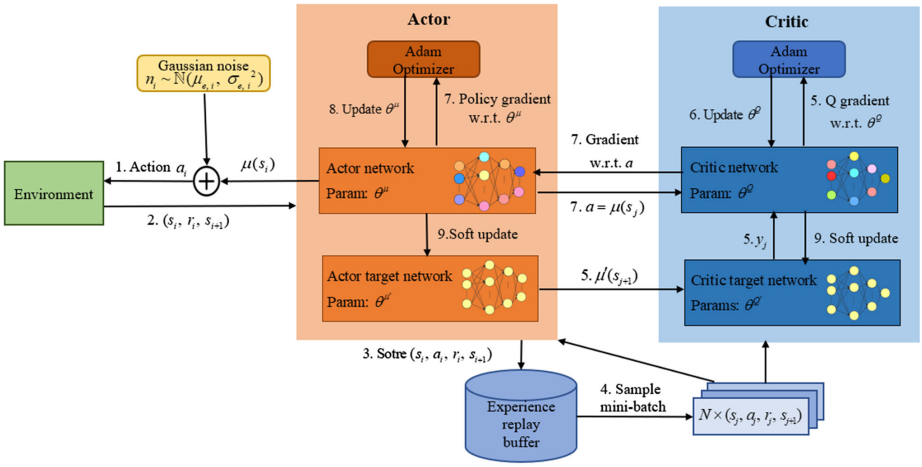
The critic network Q(s, a$|\theta^Q$) can be updates as:

$$L(\theta^Q) = E_{\mu'}[y_i - Q(s_i, a_i|\theta^Q))^2] \tag{7}$$

where $y_i = r_i + \gamma Q(s_{i+1}, \mu(s_{i+1}|\theta^Q)$.

The policy gradient $\nabla_{\theta^\mu} J \approx E_{\mu'}[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i|\theta^\mu)}]$ can be updated by using the Chain rule as follows:

$$\nabla_{\theta^\mu} J = E_{\mu'}[\nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_i}] \tag{8}$$




The complete training process for the DDPG algorithm can be summarized as follows: First, the actor network $\mu$ outputs $\mu s_i$ after the previous training step, at a certain balance between exploration and exploitation of the state space. After performing the action $a_i$, obtained by adding some Gaussian noise, the agent observes the next state $s_{i+1}$ and the immediate reward $r_i$. This transition is then stored in the experience relay buffer, from where the critic network randomly selects a min-batch of N transitions and calculates a target value for the given mini-batch. The DDPG agent then softly updates the actor and critic networks.

# 4 DDPG based algorithm

## 4.1 State Space

The state space is jointly determined by K UEs, a UAV and their environment. In a particular time slot, the remaining energy of the UAV battery, locations of the UAV and UEs, the size of generated task and the remaining task size to be executed define the system state.

## 4.2 Action Space

Based on the current state of the system and the observed environment, the agent selects actions including the UE to be served, flight angle and speed of the UAV, and task offloading ratio in the given time slot. The actor network in DDPG outputs continuous actions which help optimise the given variables, thus minimising the system cost.

## 4.3 Reward Function

The agent's behaviour is reward-based, and the choice of an appropriate reward function plays a vital role in the performance of the DDPG framework. We aim

to maximise reward by minimising the processing delay defined in the problem.

# 5 Results and Analysis

## 5.1 Parametric Analysis

We perform a series of experiments where we observe the plot for the convergence rate of the algorithm with reference to the variation in the values of the hyper-parameters such as. This way, we try to find the optimal values for the hyper parameters.

(Simulations with hyper-parameter variations:)

## 5.2 Comparison

The simulation results show that, as compared with the baseline algorithms (like DQN), our proposed DDPG algorithm can achieve better performance in terms of processing delay.

# 6 Conclusion

We showcase the proposed DDPG-based framework for computation offloading in the UAV-assisted MEC system by simulating them numerically. The simulation parameters are introduced and the performance of the DDPG-based framework is tested and validated under different circumstances, and compared with other baseline schemes. We analyze the parameters of DDPG algorithm through simulation, and compare the impacts of different parameters, including learning rate, discount factor and training strategies. The simulation results show that, as compared with the baseline algorithms, our proposed algorithm can achieve better performance in terms of processing delay.