# Distributed Task Migration Optimization in MEC by Extending Multi-Agent Deep Reinforcement Learning Approach

Chubo Liu, *Member, IEEE*, Fan Tang, Yikun Hu, Kenli Li, *Senior Member, IEEE*, Zhuo Tang, and Keqin Li, *Fellow, IEEE*

**Abstract**—Closer to mobile users geographically, mobile edge computing (MEC) can provide some cloud-like capabilities to users more efficiently. This enables it possible for resource-limited mobile users to offload their computation-intensive and latency-sensitive tasks to MEC nodes. For its great benefits, MEC has drawn wide attention and extensive works have been done. However, few of them address task migration problem caused by distributed user mobility, which can't be ignored with quality of service (QoS) consideration. In this article, we study task migration problem and try to minimize the average completion time of tasks under migration energy budget. There are multiple independent users and the movement of each mobile user is memoryless with a sequential decision-making process, thus reinforcement learning algorithm based on Markov chain model is applied with low computation complexity. To further facilitate cooperation among users, we devise a distributed task migration algorithm based on counterfactual multi-agent (COMA) reinforcement learning approach to solve this problem. Extensive experiments are carried out to assess the performance of this distributed task migration algorithm. Compared with no migrating (NM) and single-agent actor-critic (AC) algorithms, the proposed distributed task migration algorithm can achieve up 30-50 percent reduction about average completion time.

**Index Terms**—Energy, mobile edge computing, mobility, multi-agent reinforcement learning, task migration

## 1 Introduction

### 1.1 Motivation

Recently, a variety of mobile applications, such as online gaming, virtual reality (VR), and augmented reality (AR), have become more and more popular in people's daily life [1], [2]. Nevertheless, the aboved mobile applications are usually computation-intensive and latency-sensitive, thus resource-limited mobile devices face a great challenge to meet resource and latency demands from these mobile applications [3]. An alternative solution for mobile devices is offloading tasks to a cloud center which has sufficient computation resources. Unfortunately, centralized cloud computing is not suitable for these applications, due to huge latency caused by long transmission. To tackle this problem, mobile edge computing (MEC) [4] is proposed and regarded as a promising technology by deploying servers at the edge of networks. Compared to the traditional cloud computing, MEC is geographically closer to mobile users, and thus can response users faster [5]. Resource-limited mobile devices can offload their computation-intensive and latency-sensitive tasks to MEC nodes for execution to improve quality of service (QoS). For its great benefits, MEC has drawn wide attention from the academic circle and extensive works have been done. However, MEC also faces many challenges. For example, users tend to be self-decided and most of existing works address computation offloading problem with quasi-static scenario [6], and the migration management issues associated with user mobility are rarely involved.

As shown in Fig. 1, there is a sample mobile user endowed with a resource-limited device. The mobile user is moving to location 2 after offloading its task to the MEC node at location 1. In other words, mobile user may leave the coverage of the MEC node that its task is offloaded to. With QoS consideration, task migration problem caused by user mobility cannot be ignored, and efficient migration strategy that determines whether to migrate tasks dynamically to follow users' mobility should be designed to meet QoS requirement. When considering multiple independent users, the problem becomes even harder.

To our knowledge, only a few works address the task migration problem caused by distributed user mobility in MEC. Some works assume that user mobility information can be perfectly predicted [7], [8], [9], [10], [11]. However, user mobility is extremely hard to predict with perfect accuracy in realistic situations. On the other hand, some works assume that the movement of mobile users is memoryless with a sequential decision-making process, thus Markov

• *Chubo Liu, Fan Tang, Yikun Hu, Kenli Li, and Zhuo Tang are with the College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China, and also with the National Supercomputing Center in Changsha, Changsha, Hunan 410082, China. E-mail: {liuchubo, fantang, yikunhu, lkl, ztang}@hnu.edu.cn.*
• *Keqin Li is with the College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China, also with the National Supercomputing Center, Changsha, Hunan 410082, China, and the Department of Computer Science, State University of NY, New Paltz, NY 12561 USA. E-mail: lik@newpaltz.edu.*

*Manuscript received 1 July 2020; revised 23 Oct. 2020; accepted 30 Nov. 2020. Date of publication 23 Dec. 2020; date of current version 11 Feb. 2021. (Corresponding authors: Kenli Li and Keqin Li.) Recommended for acceptance by P. Balaji, J. Zhai, and M. Si. Digital Object Identifier no. 10.1109/TPDS.2020.3046737*

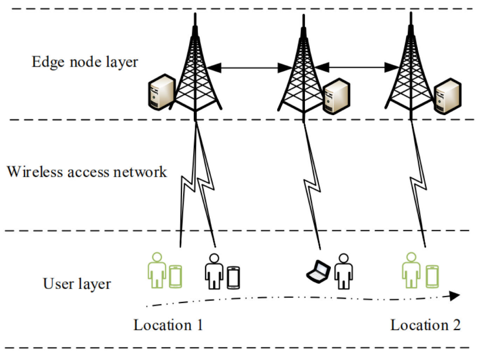


Fig. 1. System model.

Decision Process (MDP) method is applied to solve the task migration problem [12]. Nevertheless, traditional solutions such as dynamic programming require iteratively adjusting task migration decisions with high computational complexity, which is infeasible for latency-sensitive applications in MEC. With the popularity of deep reinforcement learning, some works try to apply deep reinforcement learning algorithm to task migration problem [13], [14], [15], [16]. However, the deep reinforcement learning algorithms they applied are single agent, in which the multi-task environment is unstable and the influence among multiple tasks is overlooked. Furthermore, although a beneficial migration can reduce latency of tasks, energy consumption for MEC node per migration can't be ignored. Designing an efficient task migration strategy for distributed users becomes more challenging when migration energy budget is involved. Therefore, we try to devise an efficiently distributed task migration algorithm to solve the energy-aware task migration problem based on counterfactual multi-agent (COMA) reinforcement learning approach.

## 1.2 Related Work

Some works involving user mobility in MEC are done in recent years [17]. Usually, these works can be categorized from three perspectives, i.e., mobility prediction, online processing, and Markovian hypothesis.

With the development of deep neural network, works for the first category usually assume that user mobility information is perfectly predicted. In [7], Nadembega *et al.* made a tradeoff between execution overhead and latency based on a mobility prediction method DAMP [8]. By exploiting mobility prediction information, the authors tried to reduce offloading delay by choosing a suitable communication path [9]. Wang *et al.* also predicted user mobility to minimize average latency with prediction errors consideration [10]. Aissioui *et al.* were concerned with the service migration problem of automated driving scenario by exploiting the vehicle mobility information [11]. Nevertheless, user mobility is extremely hard to predict with perfect accuracy in realistic situations [18].

Studies for the second category usually deal with issues of online user mobility, in which task migration can be triggered after the user's location has changed. Sun *et al.* addressed the mobility management problem based on Lyapunov optimization, in order to minimize the average delay within the long-term energy consumption constraint [19]. However, they only involved a single user scenario and user-centric offloading energy consumption, instead of energy consumption of MEC node for migration. Ouyang *et al.* extended it to a multi-user scenario and made a performance-cost trade off based on Lyapunov optimization similarly [20].

The movement of users is memoryless with a sequential decision-making process [12]. Therefore, studies for last category usually adopt the method of Markov Decision Process with the assumption that user mobility follows a Markovian process. Taleb *et al.* defined an analytical model with a single user case for the Follow-Me Cloud, but specific migration scheme was not involved [21]. Taleb *et al.* extended their work in [22], a traditional MDP method was adopted to trade off between migration cost and QoS, but they only discussed single user scenario with one-dimension mobility. In [23], an optimal threshold decision policy was devised to minimize time cost, similarly, single user scenario with one-dimension mobility is considered. Wang *et al.* extended their work to two-dimension mobility, and applied an improved policy iteration method to this problem with approximate solutions [24]. Nevertheless, all of the aboved works ignored migration energy consumption, in which traditional MDP methods were adopted with high computational complexity.

Therefore, some researchers tried to apply deep reinforcement learning based on Markov chain model to task migration problem in MEC, due to its low computational complexity [25], [26]. Zhang *et al.* devised a deep Q-network (DQN) [27] based algorithm for task migration, in which reward was defined as the difference between QoS and the migration cost [13]. In [14], the authors tried to minimize migration and communication cost by Q-learning based and DQN based algorithms. However, these works only considered a single user scenario. Zeng *et al.* addressed the task migration problem with multi-user scenario, and DQN based algorithm was devised to maximize the reward defined as the reciprocal of migration and communication overhead [15]. The authors in [16] formulated multiple containers migration problem as multiple dimensional MDP spaces, DQN-based deep reinforcement learning algorithm was introduced to reduce the large MDP spaces effectively. However, to our knowledge, almost all works that apply classical DQN reinforcement learning to migration problem in MEC only involves a single agent. The local state of all users is usually jointed as the global state when DQN is applied to multi-user scenario, which leads to that multi-user environment is unstable and the influence among multiple users is overlooked.

To meet average QoS under a distributed environment, a cooperative multi-agent reinforcement learning algorithm is suitable and necessary to utilize cooperation among users. Due to the above limitations, based on cooperative counterfactual multi-agent (COMA) reinforcement learning approach, we try to devise a distributed task migration algorithm to minimize the average completion time of tasks, in which migration energy consumptions of MEC nodes are also involved.

## 1.3 Contributions

In this paper, we investigate task migration problem for user mobility with multiple users scenario in MEC. The users are independent and each mobile user offloads its computation-intensive and latency-sensitive task to MEC node for execution. Moving across multiple MEC nodes, distributed users may move out side of the coverage of the MEC node that their tasks offloaded to, thus an efficiently distributed task migration algorithm is necessary to be devised with QoS consideration. Besides, migration energy consumptions of MEC nodes are also involved to make a trade off between performance and cost. Generally, the movement of mobile user is memoryless with a sequential decision-making process [12], thus reinforcement learning algorithm based on Markov chain model is applied with low computation complexity. Specifically, utilizing cooperation among users, a distributed counterfactual multi-agent (COMA) reinforcement learning approach is suitable and necessary to meet average QoS. In this paper, the goal is to minimize the average completion time of tasks under migration energy budget. Main contributions of this paper can be summarized as

- We formulate task migration problem with multiple users scenario in MEC as a minimum optimization problem with constraint, in which migration energy budget of MEC nodes is involved.
- We devise a distributed task migration algorithm based on a counterfactual multi-agent (COMA) reinforcement learning approach, taking advantage of facilitating cooperation among users with low computation complexity.
- We carry out extensive experiments to assess the performance of proposed distributed migration algorithm based on multi-agent reinforcement learning algorithm. Compared with no migrating (NM) and single-agent actor-critic (AC) algorithms, the proposed task migration algorithm can achieve up 30-50 percent reduction about average completion time.

The rest of this paper is organized as follows. Mathematical model and definitions of notations are presented in Section 2. We formulate an optimization problem in Section 3. In Section 4, we introduce the reinforcement learning setting of our system and devise a distributed task migration algorithm based on counterfactual multi-agent (COMA) reinforcement learning approach. Extensive experiments are carried out to assess the proposed COMA-based task migration algorithm in Section 5. Furthermore, conclusion and future works are presented in Section 6.

## 2 System Model

### 2.1 System Overview

In this section, we introduce the system model by defining some notations. As shown in Fig. 1, we consider that a set of $M$ MEC nodes attach base stations, denoted by $\mathcal{M} = \{1, 2, \ldots, M\}$, endowed with computation and storage resources. And there are $N$ mobile users with resource-limited devices, denoted by $\mathcal{N} = \{1, 2, \ldots, N\}$. Each mobile user offloads its computation-intensive and latency-sensitive task to MEC node for execution. With consideration of user mobility, we focus on a dynamic scenario that mobile users move across multiple MEC nodes. Thus we need to decide whether to migrate task among MEC nodes to follow user's moving trajectory. Our system operates in a time-slotted fashion $t \in \{1, 2, \ldots, T\}$, and the length of one time slot is equal to $\tau$. Specially, for mobile user $n \in \mathcal{N}$, the MEC node that executes its task $J_n$ at time slot $t$ is called serving node of user $n$, denoted by $ser_n^t \in \mathcal{M}$. On the other hand, MEC node that covers user $n$ at time slot $t$ are called connected node of user $n$, denoted by $co_n^t \in \mathcal{M}$. If $ser_n^{t-1} \neq co_n^t$, then we need to make a migration decision for mobile user $n$ at time slot $t$. Here $live_n^t = 0$ if task $J_n$ is completed before time slot $t$ and $live_n^t = 1$ otherwise. Migration decisions of all tasks at time slot $t$ are denoted by $\mathcal{X}^t = \{x_1^t, x_2^t, \ldots, x_n^t\}$, where $x_n^t \in \mathcal{M}$. Specially, $x_n^t = co_n^t$ and $x_n^t = ser_n^{t-1}$ denote that task $J_n$ will be or not be migrated from $ser_n^{t-1}$ to $co_n^t$ at time slot $t$, respectively. The notations we used are summarized in Table 1. We next introduce communication model, computation model, and migration model in details.

TABLE 1
Notations

| Notation | Definition |
|---|---|
| $M, N$ | number of MEC node, number of mobile user |
| $f_m$ | maximal CPU frequency of MEC node $m$ |
| $\varphi_m^t$ | number of tasks executed on MEC node $m$ |
| $J_n$ | task of mobile user $n$ |
| $\lambda_n, o_n$ | input data size and output data size of task $J_n$ |
| $\gamma_n$ | workload requirement of task $J_n$ |
| $\psi_n^t$ | remaining CPU cycles requirement at time slot $t$ |
| $p_n$ | transmission power of mobile user $n$ |
| $\omega_m$ | wireless bandwidth of MEC node $m$ |
| $\varrho^m$ | white noise power |
| $H_{m,n}$ | channel gain between MEC node $m$ and user $n$ |
| $x_n^t$ | MEC node that executes task $J_n$ at time slot $t$ |
| $ser_n^t$ | Serving MEC node of user $n$ at time slot $t$ |
| $co_n^t$ | Connected MEC node of user $n$ at time slot $t$ |
| $r(m,n)$ | transmission rate between node $m$ and user $n$ |
| $T_u(m,n)$ | transmission time between node $m$ and user $n$ |
| $r_{mm}$ | data transmission rate among MEC nodes |
| $T_e(n)$ | remaining execution time of task $J_n$ in time slot $t$ |
| $T_m^n$ | total migration time of task $J_n$ |
| $E$ | the energy consumption per bit for task migration |
| $E_{budget}$ | the energy consumption budget for migration |
| $\tau$ | length of time slot |
| $D(n)$ | completion time of task $J_n$ |

### 2.2 Communication Model

In this paper, we use a general parameter model $J_n \triangleq (\lambda_n, \gamma_n, O_n)$ to describe task, in which $\lambda_n$ (in bits) is input data size of task $J_n$, $\gamma_n$ (in CPU cycles/bit) denotes how many CPU cycles are required per bit for task $J_n$, and $O_n$ is output data size of task $J_n$. Transmission time of output data is not ignored, and we assume that downlink data transmission rate is equal to uplink data transmission rate. Based on a general communication model, data transmission rate between MEC node $m$ and mobile user $n$ can be calculated as

$$r(m,n) = w_m \log_2\left(1 + \frac{p_n H_{m,n}}{\varrho^m}\right), \tag{1}$$

where $w_m$ is the channel bandwidth that MEC node $m$ allocates for mobile user $n$, $p_n$ is transmission power of mobile user $n$, $H_{m,n}$ is channel gain, $\theta^m$ is white noise power. Thus transmission time of input data and output data can be expressed as

$$T_u(n) = \frac{\lambda_n}{r(m,n)} + \frac{O_n}{r(m',n)}, \tag{2}$$

where $m$ denotes MEC node that mobile user $n$ first offloads its task to, and $m'$ denotes MEC node that connects with mobile user $n$ when its task $J_n$ is completed.

### 2.3 Computation Model

As mentioned before, each mobile user with a resource-limited device offloads its task to MEC node for execution. We assume that each MEC node is able to process multiple tasks simultaneously by sharing processor. Specifically, $f_m$ and $\varphi_m^t$ denote the maximum computation capability (i.e., CPU cycles per second) of MEC node $m$ and number of tasks executed on MEC node $m$ at time slot $t$. Then processing time of task $J_n$ executed on MEC node $m$ at time slot $t$ can be estimated as

$$T_e(n) = \frac{\psi_n^t \varphi_m^t}{f_m}, \tag{3}$$

$$ser_n^t = m, \quad \forall n, t, m, \tag{4}$$

where $\psi_n^t$ denotes how many remaining CPU cycles are required for task $J_n$ at the beginning of time slot $t$, $ser_n^t$ is the serving MEC node of mobile user $n$ at time slot $t$.

### 2.4 Migration Model

To satisfy QoS, task migration problem caused by user mobility is necessary to be addressed. When serving node $ser_n^{t-1}$ of last time slot $t-1$ is different from the connected node $co_n^t$ of task $J_n$, then we need to make a task migration for it. If task $J_n$ is decided to be migrated from serving node $ser_n^{t-1}$ to connected node $co_n^t$, then migration time of input data can't be ignored. We assume that whole input data of task needs to be transmitted from $ser_n^{t-1}$ to $co_n^t$ if there is a migration. Thus we can calculate migration time of task $J_n$ as

$$\begin{aligned} T_m(n) = &\sum_{t=1}^{t=T} I\{ser_n^t = co_n^t\} \frac{\lambda_n}{r_{mm}} \\ &+ I\{live_n^t = 0 \quad and \quad ser_n^t \neq co_n^t\} \frac{O_n}{r_{mm}}, \end{aligned} \tag{5}$$

where first part denotes the sum of migration time of input data for each migration. Here $I\{x\} = 1$ if the event $x$ is true and $I\{x\} = 0$ otherwise. And $r_{mm}$ is data transmission rate among MEC nodes. The second part denotes migration time of output data if serving node $ser_n^t$ is different with connected node $co_n^t$ of task $J_n$, when task $J_n$ is completed.

It is intuitive that completion time of task can be reduced when it is decided to be migrated. However, from the prospective of MEC nodes, there is non-negligible energy consumption when each migration occurs. We use $E$ to denote the energy consumption per bit for task migration. Thus the total migration energy consumption for all mobile users can be calculated as

$$\begin{aligned} E_{total} = &\sum_{n=1}^{n=N} \sum_{t=1}^{t=T} I\{x_n^t = co_n^t\} \lambda_n E \\ &+ I\{live_n^t = 0 \quad and \quad ser_n^t \neq co_n^t\} O_n E, \end{aligned} \tag{6}$$

where $\lambda_n E$ is the migration energy consumption of task $J_n$ once, and $\sum_{t=1}^{t=T} I\{x_n^t = co_n^t\}$ denotes the total number of migration. $I\{x_n^t = 0 \quad and \quad ser_n^t \neq co_n^t\} O_n E$ denotes migration energy consumption of output data if serving node $ser_n^t$ is different with connected node $co_n^t$ of task $J_n$, when task $J_n$ is completed.

## 3 Problem Formulation

We have discussed computation, communication, and migration model in the last section. As mentioned earlier, $live_n^t = 0$ means that task $J_n$ is completed before time slot $t$. Then completion time of task $J_n$ can be obtained by the sum of time slots when $live_n^t > 0$,

$$D(n) = \sum_{t=1}^{t=T} I\{live_n^t > 0\}\tau, \tag{7}$$

where $\tau$ is the length of one time slot.

Note that we try to make a migration decision at each time slot to minimize the average completion time of all tasks, while the total energy consumption is under the energy budget $E_{budget}$. Mathematically, our optimization problem can be formulated as

$$\text{minimize} \qquad \frac{\sum_{n=1}^{n=N} D(n)}{N}, \tag{8}$$

$$\text{s.t.} \qquad E_{total} \leq E_{budget}, \tag{9}$$

$$x_n^t \in \mathcal{M} \quad \forall n, t. \tag{10}$$

Unfortunately, we find the optimization problem is NP-Hard. Next, proof of NP-hardness for this problem is presented.

**Theorem 1.** The optimization problem is NP-hard.

**Proof.** To proceed, we first introduce the knapsack problem (KP), which is a well-known NP-hard problem. KP: Given $N$ items with sizes $v_i$ and profit values $p_i$ for $i \in \mathcal{N}$ and a knapsack with size $V$, the objective is to find a set of items $\mathcal{S} \subseteq \mathcal{N}$ to pack such that the total values of knapsack is maximized, mathematically, the knapsack problem can be formulated as

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in \mathcal{S}} p_i, \\ \text{s.t.} \quad & \sum_{i \in \mathcal{S}} v_i \leq V. \end{aligned}$$

Here we consider a special instance of our optimization problem, in which there are $N$ tasks $J_1, J_2, \ldots, J_n$ with the migration energy budget $E_{budget}$. The mobile users moves across multiple MEC nodes. With different migration schemes for each time slot, each task may be executed on serving MEC node or migrated to connected MEC node for execution. We use $t_i$ to denote completion time of task $J_i$, and $E_i$ to denote migration energy consumption for task $J_i$. Then, our optimization problem can be formulated as

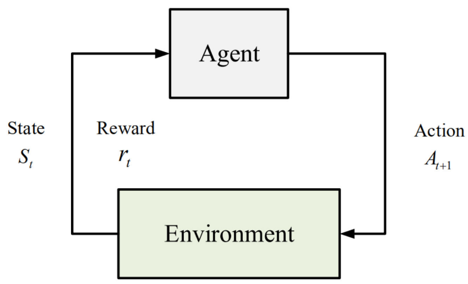


Fig. 2. Formal description of reinforcement learning.

$$\begin{aligned} \text{minimize} \quad & \frac{\sum_{i=1}^{i=N} t_i}{N}, \\ \text{s.t.} \quad & \sum_{i\in\mathcal{N}} E_i \le E_{budget}. \end{aligned}$$

For each task $J_i$, we define $q_i = -\frac{t_i}{N}$, then the optimization problem can be formulated as

$$\begin{aligned} \text{maximize} \quad & \sum_{i\in\mathcal{N}} q_i, \\ \text{s.t.} \quad & \sum_{i\in\mathcal{N}} E_i \le E_{budget}. \end{aligned}$$

This special instance of our problem corresponds to the KP with knapsack size $V = E_{budget}$ and profit value $p_i = q_i$. Therefore, this problem is NP-hard. This completes the proof. □

As is presented before, the movement of mobile users is memoryless with a sequential decision-making process [12], thus MDP method can be applied. Nevertheless, traditional solutions such as dynamic programming require iteratively adjusting task migration decisions with high computational complexity, which is infeasible for latency-sensitive applications in mobile edge computing. Therefore, we adopt deep reinforcement learning algorithm based on Markov chain model to tackle this task migration problem with low computational complexity.

## 4 Distributed Task Migration Algorithm

We have formulated the task migration problem as a minimum optimization problem before. In this section, a distributed task migration algorithm based on counterfactual multi-agent (COMA) reinforcement learning approach, is devised to solve the task migration problem.

### 4.1 Reinforcement Learning Settings

As shown in Fig. 2, main idea of reinforcement learning algorithm is that agent makes better decisions by constantly interacting with the environment. For each time slot, agent collects state $S_t$ of system and evaluates the reward $r_t$ for environmental feedback of last action $A_{t-1}$, then agent makes a better action $A_t$ to act on the environment.

In this paper, formal definition of our system for the above-mentioned elements in reinforcement learning model are as follows.

- Agent: MEC node controller is the agent of our system. The MEC node controller receives information of MEC nodes and tasks, and location information of mobile users, then it makes task migration decisions for all tasks at each time slot.
- State: in our system, state is defined as joint information of all tasks. Specifically, state $S_t$ at time slot $t$ can be expressed as

$$S_t = \{W_1^t \times W_2^t \times \dots W_N^t\}, \tag{11}$$

$$W_n^t = \{\psi_n^t, ser_n^{t-1}, co_n^t\}, n \in \{1, 2, \dots, N\}, \tag{12}$$

where $\psi_n^t$ denotes how many remaining CPU cycles are required for task $J_n$ at the beginning of time slot $t$, $ser_n^{t-1}$ is the serving node that processes task $J_n$ at time slot $t-1$, and $co_n^t$ denotes the connected node that covers mobile user $n$.
- Action: as we said before, agent makes a better action $A_t$ for each task by constantly interacting with the environment. $A_t = \{x_1^t, x_2^t, \dots, x_n^t\}$, which means action that agent makes at time slot $t$ is joint migration decisions for all tasks. Specially, $x_n^t = co_n^t$ and $x_n^t = ser_n^{t-1}$ denote that task $J_n$ will be or not be migrated from serving node $ser_n^{t-1}$ to connected node $co_n^t$ for execution at time slot $t$, respectively.
- Reward: at time slot $t$, agent evaluates the reward $r_t$ based on environmental feedback of last action $A_{t-1}$. In our system, we define reward $r_t$ at time slot $t$ as the difference between average estimated completion time $Avg_{t-1}$ for all tasks at last time slot and $Avg_t$ at current time slot,

$$r_t = Avg_{t-1} - Avg_t, \tag{13}$$

$$\begin{aligned} Es_n^t = \tau(t-1) &+ \frac{\psi_n^t}{\frac{f_{ser_n^t}}{\sum_{i\in\mathcal{N}} I\{x_i^t = x_n^t\}}} \\ &+ I\{ser_n^t = co_n^t\}\frac{\lambda_n}{r_{mm}} \\ &+ I\{live_n^t = 0 \quad and \quad ser_n^t \ne co_n^t\}\frac{O_n}{r_{mm}}, \end{aligned} \tag{14}$$

$$Avg_t = \frac{\sum_{n=1}^{n=N} Es_n^t}{N}, \tag{15}$$

where $Es_n^t$ is the estimated completion time of task $J_n$ at time slot $t$, including the sum of past time slots$\tau(t-1)$ for processing task $J_n$, the second part $\frac{\psi_n^t}{\frac{f_{ser_n^t}}{\sum_{i\in\mathcal{N}} I\{x_i^t = x_n^t\}}}$ denotes the remaining processing time of task $J_n$ with the assumption that task $J_n$ is executed on serving node $ser_n^t$ until completed, the third part $I\{ser_n^t = co_n^t\}\frac{\lambda_n}{r_{mm}}$ is the migration time of input data if task $J_n$ is migrated from $ser_n^{t-1}$ to $co_n^t$ at time slot $t$, the last part $I\{live_n^t = 0 \quad and \quad ser_n^t \ne co_n^t\}\frac{O_n}{r_{mm}}$ denotes the migration time of output data if task $J_n$ is completed at time slot $t$ and its serving node is different from connected node. The total reward is the sum of reward at each episode

($r = r_1 + r_2 + \ldots + r_T$), which is the key evaluation metrics of reinforcement learning algorithm.

Q-learning is a classical reinforcement learning algorithm, in which each state-action pair is evaluated by Q-value stored in Q-matrix. However, Q-matrix is huge and storage-consumed with the large size, due to high dimensional of state and action. To tackle this issue, deep neural networks (DNN) is introduced to estimate the Q-value of state-action pair based on traditional Q-learning method, that is deep Q-network (DQN) algorithm [27]. And there are some improved reinforcement learning algorithms, such as Policy Gradients [28], DDPG [29], A3C [30] and so on, however these reinforcement learning algorithms are single-agent. It is not suitable to apply these single-agent reinforcement learning algorithms to multi-task problem in mobile edge computing. The reason lies in that the local state of all users is usually jointed as the global state when single-agent reinforcement learning approaches are applied to multi-user scenario, which leads to that multi-user environment is unstable and the influence among multiple users is overlooked. Therefore, we introduce a counterfactual multi-agent deep reinforcement learning approach to our system, COMA, considering cooperation among tasks to achieve the global goal.

## 4.2 Actor-Critic Algorithm

Counterfactual multi-agent (COMA) policy gradients is a class of actor-critic reinforcement learning approach [28]. Here are some necessary backgrounds need to be presented before introducing the original actor-critic algorithm. The above mentioned Q-learning is a value-based algorithm, which chooses action based on approximating the value of state-action $Q(S,A)$, such as choosing the action whose value is maximal on current state $\pi(S) = \underset{A}{\operatorname{argmax}} \quad Q(S,A)$.

**Algorithm 1.** Policy Gradient Algorithm

```
Initialize the parameter $\theta$ randomly
for each episode $\{S_1, A_1, r_1, \ldots, S_{T-1}, A_{T-1}, r_{T-1}\}$ do
  for $t = 1$ to $t = T-1$ do
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(S_t, A_t) V_t$;
  end for
end for
return $\theta$;
```

Policy gradient [29] is a policy-based algorithm, which outputs directly the probability of each action for current state based on neural network. Specifically, the parameter $\theta$ updating process of neural network is shown in Algorithm 1. Loss is defined as $\log \pi_\theta(S_t, A_t) V_t$, in which $\pi_\theta(S,A) = P(A|S,\theta)$ means the probability of choosing action $A$ for current state $S$ when parameter of neural network is $\theta$. And $V_t$ can be the total reward or the value of state-action. Thus in policy gradient algorithm, if an action receives more reward, the probability of its occurrence will be increased, otherwise, if an action receives less reward, we decrease the probability of its occurrence. Note that policy gradient algorithm is episode-updated instead of step-updated, which makes a low learning efficiency.

Frame of actor-critic approach is shown in Fig. 3. Actor-critic approach is consisted of an actor and a critic. Actor is a policy neural network based on policy gradient algorithm. Critic is a value-based neural network which estimates the value of state. For each time slot $t$, actor outputs an action $A_t$ to act on the environment, then critic evaluates the reward for environmental feedback of action $A_t$, and calculates the TD error $\delta_t$ to guide the parameter updating of actor network. Specifically, the TD error $\delta_t$ is given by

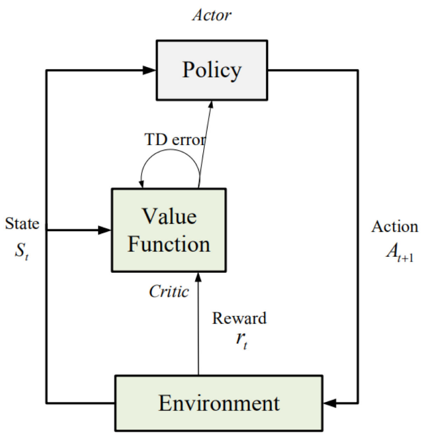


Fig. 3. Frame of actor-critic approach.

$$\delta_t = r_{t+1} + \gamma V(S_{t+1}) - V(S_t), \tag{16}$$

$$V(S) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = S\right], \tag{17}$$

where $V(S)$ is the value of state $S$, $\gamma$ is the discount factor. Parameter $\theta_{actor}$ updating of actor network is guided by TD error, $\theta_{actor} \leftarrow \theta_{actor} + \alpha \nabla_{\theta_{actor}} \log \pi_{\theta_{actor}}(S_t, A_t)\delta_t$. On the other hand, loss of critic network is defined as the square of TD error.

## 4.3 COMA-Based Distributed Task Migration Algorithm

When faced with task migration problem with multiple users, we intuitively think that each agent learn independently is simplest way to apply policy gradients with multiple agents consideration. And each agent of user has its own actor and critic to learn with user's own state-action information. This is independent actor-critic (IAC), which is inspired by popular multi-agent learning algorithm independent Q-learning [30]. Although it is simple to understand and implement, it is hard to learn coordinated strategies due to lack of interactions between multiple agents, or evaluate the contribution of action of an individual agent to the global reward, or fail to exploit the fact that learning is centralised [31]. Therefore, counterfactual multi-agent (COMA) policy gradients [32] is introduced to tackle these issues.

Main characteristics of COMA can be summarized as introducing a centralised critic and a counterfactual baseline. As is shown in Fig. 4. To exploit the fact that learning is centralised, COMA introduces a centralised critic. Each actor conditions on its own state histories of corresponding user, however, input of the centralised critic is the joint state of each user. As we said in last subsection, in single-agent scenario, parameter $\theta_{actor}$ updating of actor network is

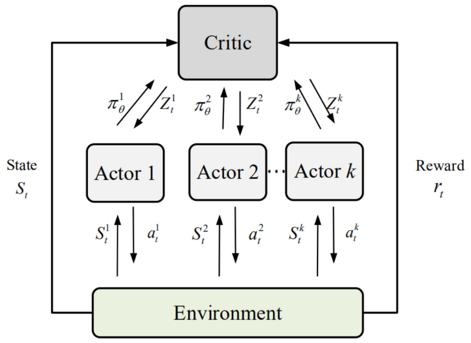

Critic
$\pi_\theta^1$ $Z_t^1$ $\pi_\theta^2$ $Z_t^2$ $\pi_\theta^k$ $Z_t^k$
State $S_t$
Actor 1
Actor 2
Actor $k$
Reward $r_t$
$S_t^1$ $a_t^1$ $S_t^2$ $a_t^2$ $S_t^k$ $a_t^k$
Environment

Fig. 4. Distributed framework of COMA.

usually based on TD error calculated by critic. Nevertheless, it is not suitable to use TD error to update parameter $\theta_{actor}$ of actor network in multi-agent reinforcement learning algorithm. The reason lies in that TD error considers only global reward, and each actor does not explicitly to know how its actions contribute to the global reward.

Therefore, COMA introduces a counterfactual baseline to solve this problem. Centralised critic estimates Q-values $Q(S_t, A_t)$ of joint action $A_t = \{a_t^1, a_t^2, \ldots, a_t^n\}$ and state $S_t = \{S_t^1, S_t^2, \ldots, S_t^n\}$. For each agent $k$, keeping actions of agents $A_t^{-k}$ fixed except agent $k$, the advantage function that compares the Q-value can be calculated as

$$Z_t^k(S_t, A_t) = Q(S_t, A_t) - \sum_{a_t^k} \pi^k(a_t^k | S_t^k) Q(S_t, (A_t^{-k}, a_t^k)). \tag{18}$$

Hence, $Z_t^k(S_t, A_t)$ computes a baseline for agent $k$ to present how its actions contribute to the global reward based on centralised critic. In conclusion, COMA is aimed at maximizing the global reward by cooperation of multiple agents. Thus it is reasonable to apply COMA to our task migration problem with global goal that minimizing the average completion time of all tasks. Each actor is learned for each user's task in a distributed way, which chooses action that determines the task whether migrate or not. The centralised critic calculates contribution of each task to the global goal, then guides the learning of each actor. Next, specific implementation steps of COMA-based distributed task migration algorithm is summarized as Algorithm 2.

In Algorithm 2, we initialise network with random parameter (Step 1). For centralised critic, a double network structure is used to reduce the correlation between the current Q value and the target Q value and improve the stability of the algorithm. In addition, to speed learning, an actor network is reused for all agents by sharing parameters. The network is trained in batch mode. For each training epoch $i$, each actor with $\theta_{actor}^i$ chooses migration action for each task, and generated episodes are appended to buffer memory (Steps 3-10). $terminal$ is exit state of our system, when all tasks are completed. Unrolling states, actions and rewards in buffer memory $B$, target Q value $y_t^{(\lambda)}$ is calculated with target critic network using $\hat{\theta}_{critic}^1$ (Steps 11-16). $n$-step returns $G_t^{(n)}$ are calculated with bootstrapped values estimated by target network. Critic network is trained by minimizing loss $(y_t^{(\lambda)} - Q(S_t, A_t))^2$ (Steps 17-22), and $\hat{\theta}_{critic}^i$ is reset as $\hat{\theta}_{critic}^i = \theta_{critic}^i$ every $C$ steps. Then calculating advantage function $Z_t^k(S_t, A_t)$ for each agent $k$ based on Eq. (18), and parameter updating of actor network is guided by advantage function (Steps 23-28).

**Algorithm 2.** COMA-based Distributed Task Migration Algorithm

```
1: Initialise critic network with random parameter $\theta_{critic}^1$; Initialise target critic network with parameter $\hat{\theta}_{critic}^1 = \theta_{critic}^1$; Initialise actor network with random parameter $\theta_{actor}^1$; Initialise buffer memory $B$ to capacity $BatchSize$.
2: for each training epoch $i$ do
3:   Empty buffer memory;
4:   for (episode $e$ from 1 to $BatchSize$ ) do
5:     while $S_t \neq terminal$ and $t < T$ do
6:         For current state $S_t$, each actor with $\theta_{actor}^i$ chooses migration action for each task;
7:       Get reward $r_t$, $t = t + 1$;
8:     end while
9:     Append episode $e$ to buffer memory $B$;
10:  end for
11:  for (time slot $t$ from 1 to $T$ ) do
12:    Unroll states, actions and rewards in buffer memory $B$;
13:    Calculate $TD(\lambda)$ target Q value $y_t^{(\lambda)}$ using $\hat{\theta}_{critic}^1$;
14:    $y_t^{(\lambda)} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$;
15:    $G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V(S_{t+n})$;
16:  end for
17:  for (time slot $t$ from T to 1 ) do
18:    Unroll states, actions and rewards in buffer memory $B$;
19:    $\Delta\theta_{critic} = \nabla_{\theta_{critic}} \left(y_t^{(\lambda)} - Q(S_t, A_t)\right)^2$;
20:    $\theta_{critic}^{i+1} = \theta_{critic}^i - \alpha\Delta\theta_{critic}$;
21:    Every $C$ steps reset $\hat{\theta}_{critic}^i = \theta_{critic}^i$;
22:  end for
23:  for (time slot $t$ from T to 1 ) do
24:    Process all agents in parallel via single batch;
25:    Calculate advantage function $Z_t^k(S_t, A_t)$ for each agent $k$ based on Eq. (18);
26:    $\Delta\theta_{actor} += \alpha\nabla_{\theta_{actor}} \log \pi_{\theta_{actor}}(S_t^k, a_t^k) Z_t^k(S_t, A_t)$;
27:  end for
28:  $\theta_{actor}^{i+1} = \theta_{actor}^i + \alpha\Delta\theta_{actor}$;
29: end for
```

The actor network is based on recurrent neural network, consisting of 128-bit gated recurrent units (GRUs) that use fully connected layers both to process the input and to produce the output values from the hidden state. The critic network is based on a feed forward network with multiple ReLU layers combined with fully connected layers. Training is performed in batch mode, with a batch size of 30. For one episode, a gradient step is applied to train the feed-forward critic for each time step. The recurrent part of the actor is fully unrolled and gradients are aggregated in the backward pass across all time steps, then a gradient update is applied to train actor. In addition, a target network for the critic is applied, which updates every 150 training steps for the feed-forward critics. The actor and the critic networks are trained using RMSprop. The learning rate of actor network is 0.0001 and the learning rate of critic network is 0.001.

The convergence analysis of COMA-based task migration algorithm is given in the following theorem. With same assumptions, proof of this theorem is based on the convergence of single-agent actor-critic algorithm [28].

**Theorem 2.** *The COMA-based distributed task migration algorithm is convergent, that is*

$$\lim \inf_i \|\nabla G_i\| = 0 \quad w.p.1. \tag{19}$$

**Proof.** The gradient of COMA-based task migration algorithm is given by

$$G_i = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_k \nabla_{\theta_i} \log \pi^k\left(a^k|S^k\right) Z^k(S, A)\right], \tag{20}$$

where advantage function $Z^k(S, A)$ is defined as Eq. (18), and we define the counterfactual baseline $b\left(S, A^{-k}\right)$ as

$$b\left(S, A^{-k}\right) = \sum_{a^k} \pi^k\left(a^k|S^k\right) Q\left(S, \left(A^{-k}, a^k\right)\right), \tag{21}$$

thus the gradient of COMA-based task migration algorithm can be written as

$$G_i = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_k \nabla_{\theta_i} \log \pi^k\left(a^k|S^k\right)\left(Q(S, A) - b\left(S, A^{-k}\right)\right)\right], \tag{22}$$

where $\theta$ are the parameters of all actor networks $\theta = \left\{\theta^1, \theta^k, \ldots, \theta^N\right\}$.

First, we consider the counterfactual baseline $b\left(S, A^{-k}\right)$ for the gradient $G_i$ of COMA-based task migration algorithm,

$$G_b = -\mathbb{E}_{\boldsymbol{\pi}}\left[\sum_k \nabla_{\theta_i} \log \pi^k\left(a^k|S^k\right) b\left(S, A^{-k}\right)\right], \tag{23}$$

where $E_\pi$ is the state-action expected distribution with regard to joint policy $\pi$. In addition, as defined by [29], $d^\pi(S)$ is the stationary distribution of states under joint policy $\pi$,

$$d^\pi(S) = \lim_{t\to\infty} \Pr\{S_t = S|S_0, \pi\}. \tag{24}$$

Then the gradient $G_b$ of counterfactual baseline can be written as

$$\begin{aligned} G_b = -\sum_S d^\pi(S) \sum_k \sum_{A^{-k}} \pi\left(A^{-k}|S^{-k}\right) \cdot \\ \sum_{a^k} \pi^k\left(a^k|S^k\right) \nabla_\theta \log \pi^k\left(a^k|S^k\right) b\left(S, A^{-k}\right) \end{aligned} \tag{25}$$

$$\begin{aligned} = -\sum_S d^\pi(S) \sum_k \sum_{S^{-k}} \pi\left(A^{-k}|S^{-k}\right) \cdot \\ \sum_{a^k} \nabla_\theta \pi^k\left(a^k|S^k\right) b\left(S, A^{-k}\right) \end{aligned} \tag{26}$$

$$= -\sum_S d^\pi(S) \sum_k \sum_{A^{-k}} \pi\left(A^{-k}|S^{-k}\right) b\left(S, A^{-k}\right) \nabla_\theta 1 = 0. \tag{27}$$

According to the following equations, the counterfactual baseline $b\left(S, A^{-k}\right)$ does not change the expected gradient $G_i$, thus we can draw the conclusion that counterfactual baseline for each agent has no effect on the convergence of COMA-based task migration algorithm.

Second, we consider the remainder of $G_i$ except gradient $G_b$ of counterfactual baseline,

$$G_Q = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_k \nabla_\theta \log \pi^k\left(a^k|S^k\right) Q(S, A)\right] \tag{28}$$

$$= \mathbb{E}_{\boldsymbol{\pi}}\left[\nabla_\theta \log \prod_k \pi^k\left(a^k|S^k\right) Q(S, A)\right] \tag{29}$$

$$= \mathbb{E}_{\boldsymbol{\pi}}[\nabla_\theta \log \boldsymbol{\pi}(A|S) Q(S, A)]. \tag{30}$$

Refer to [28], standard policy gradient $G$ of a single-agent actor-critic algorithm is given by

$$G = \mathbb{E}_{\boldsymbol{\pi}}[\nabla_\theta \log \boldsymbol{\pi}(A|S) Q(S, A)], \tag{31}$$

and the proof of convergence for a single-agent actor-critic is presented on their work, with the following assumptions,

- The policy $\pi$ is differentiable.
- The learning rates for actor network and critic network are sufficiently small.
- The learning rate of actor network is sufficiently smaller than the learning rate of critic network. These assumptions are satisfied in our COMA-based task migration algorithm. Note that centralised critic network of COMA-based task migration algorithm is essential for this proof to hold. Therefore, this achieves the proof of convergence of COMA-based task migration algorithm. □

## 5 Performance Evaluation

In this section, numerical studies are conducted to evaluate performance of COMA-based distributed task migration algorithm.

### 5.1 Parameter Configuration

As shown in Table 2, there are 60 mobile users and 16 MEC nodes in our system. For MEC node $m$, CPU frequency $f_m$ of which is uniformly selected from $[0.1, 1]$ GHz, and dynamic value of which is from $[0.1, 0.5]$ GHz to $[0.5, 0.9]$ GHz. Corresponding datasets, including training data and validation data, are randomly generated to evaluate the performance of algorithm. Specifically, the mobility trajectories of mobile users are based on random walk model. The parameter settings of task workloads is general, refer to some related work. Training data and validation data are completely independent and they are separated into 4:1 based on hold-out method. Mobile users are moving randomly across multiple MEC nodes and trajectories of them are generated based on random walk model [9], [19]. Each mobile user is associated with a computation-intensive and latency-sensitive task that is offloaded to MEC node for execution. For task $J_n$, fixed value of input data size $(\lambda_n)$ is uniformly selected from $[100, 500]$ KB, and dynamic value of which is from $[100, 250]$ KB to $[100, 700]$ KB. Similarly, fixed

TABLE 2
Experiment Parameters

| Parameters | (Fixed)–[Varied range] (Increment) |
|---|---|
| Number of mobile user $(N)$ | 60 |
| Number of MEC node $(M)$ | 16 |
| Input data size $(\lambda_n)$ | [100,500]–[100, 250-700] (150) KB |
| Output data size $(O_n)$ | [100,150]–[100, 150-600] (150) KB |
| Workload $(\gamma_n)$ | [800,2400]–[800, 1600-6400] (1600) cycles/bit |
| MEC CPU frequency $(f_m)$ | [0.1,1]–[0.1-0.5, 0.5-0.9](0.1) GHz |
| Migration energy budget $E_{budget}$ | [250]–[50, 450] (100) GJ |
| Length of time slot $\tau$ | 5 s |
| Wireless bandwidth $\omega_m$ | 1 Mbps |
| Channel gain $H_{m,n}$ | $10^{-6}$ |
| Transmission power $p_n$ | 1 W |
| White noise power $\varrho^m$ | $10^{-9}$ W |

value of output data size $(O_n)$ is uniformly selected from $[100, 150]$ KB, and dynamic value of which is from $[100, 150]$ KB to $[100, 600]$ KB. For workload $(\gamma_n)$ of task $J_n$, fixed value of which is $[800, 2400]$ cycles/bit, and dynamic value of which is from $[800, 1600]$ cycles/bit to from $[800, 6400]$ cycles/bit. Fixed value of migration energy budget $(E_{budget})$ is 250 GJ, and dynamic value of which is from 50 GJ to 450 GJ. Besides, configurations of other variables are summarized in Table 2.

## 5.2 Performance Benchmark

To our knowledge, none of works applies multi-agent reinforcement learning algorithm to task migration problem in mobile edge computing. And related work mentioned is not suitable for comparison, because characteristics and performance metrics of them are different. Hence, two heuristics algorithms are introduced to evaluate the performance of our COMA-based task migration algorithm. 1) *Not Migrating (NM)*: Tasks are executed on MEC node that they are first offloaded to until they are completed, no matter where mobile users that they belongs is moving to. 2) *Actor-Critic (AC)*: The single-agent actor-critic reinforcement learning algorithm we presented in Section 4.2 is also introduced for comparison, in which inputs of actor network and critic network are joint state and action of all tasks, and the cooperation among multiple tasks toward a global goal is ignored.

## 5.3 Random Results

In our experiments, we randomly generate 100 files of data with 60 mobile users. Three of which are chose as samples to show dynamics of cumulative reward with increased episode steps in Fig. 5. As we presented before, reward $r_t$ in our system is define as the difference between average estimated completion time $Avg_{t-1}$ for all tasks at last time slot and $Avg_t$ at current time slot. In early episode steps, cumulative reward is slightly oscillating due to exploration in reinforcement learning. Then cumulative reward is tend to be stable with increased episode steps.

In the following experiments, the average completion time of all tasks is evaluated as performance metrics. Furthermore, average value of performance metrics for 100 data files is presented to make the results more convincing.

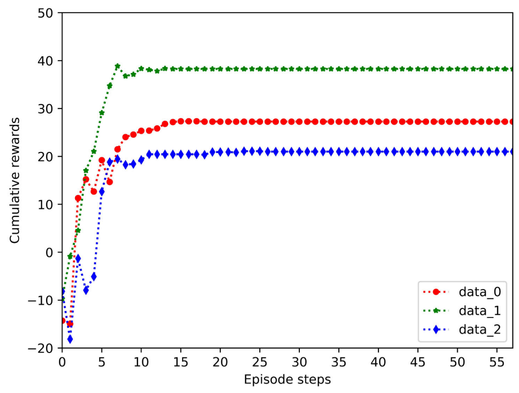


Fig. 5. Dynamic cumulative reward with three representative data files.

With other variables remain unchanged, we first increase input data sizes of tasks with an increment 150 KB. As shown in Fig. 6, we can observe that the average completion time of all tasks is increasing with increased input data sizes of tasks. That's because input data sizes of task is determining factor of computation time that increases with increased input data sizes of tasks. Nevertheless, compared to NM and AC algorithms, COMA-based distributed task migration algorithm can achieve up 30-50 percent reduction in average completion time.

In addition, impact that changes output data sizes of tasks on the average completion time of all tasks is shown in Fig. 7. Similarly, increment of output data sizes of tasks is also 150 KB. With the increased output data sizes of tasks, we can observe that there's almost no change in value of the average completion time of all tasks in Fig. 7a. The reason lies in that output data sizes of tasks can only have an effect on communication time of tasks. And the value of output data sizes of tasks is small. Thus we increase output data sizes of tasks by 100 times. The experimental results are shown in Fig. 7b. We can observe that the average completion time of all tasks is increasing with increased output data sizes of tasks. That's because migration time defined in Eq. (5) is increased, due to increased output data sizes of task. However, COMA-based distributed task migration algorithm in this paper is always outperforming NM and AC algorithms with the variation of output data size.

Keeping other variables constant, influence of the variation of workload on the average completion time of all tasks is shown in Fig. 8. Workload requirements of tasks is increased from $[800, 1600]$ to $[800, 6400]$ cycles/bit. From Fig. 8, the average completion time of all tasks is increased with the increase of workload requirements. That's because

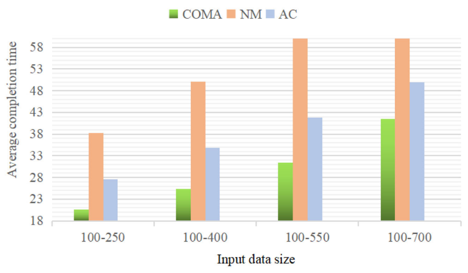
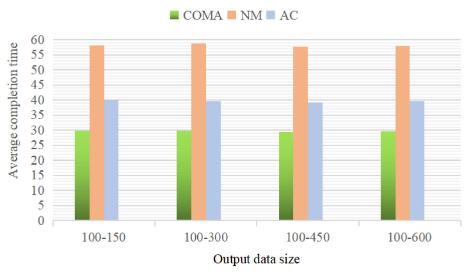
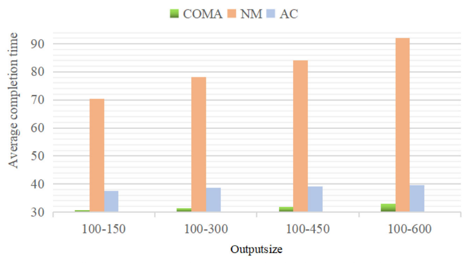


Fig. 6. The average completion time of all tasks with the variation of input data size.



(a) output from 100-600KB



(b) output from 10000-60000KB

Fig. 7. The average completion time of all tasks with increased output data size.

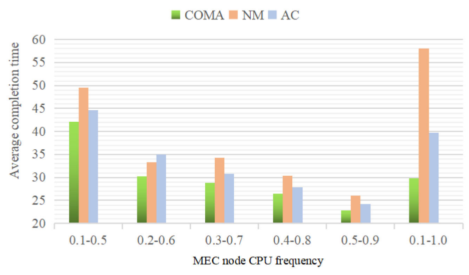


Fig. 9. The average average completion time of all tasks with varied CPU frequencies of MEC nodes.

workload requirement of task is determining factor of computation time and number of MEC nodes and CPU frequencies of MEC nodes remain unchanged. Thus the average completion time of all tasks is increased. However, as the picture shows, COMA-based distributed task migration algorithm always can achieve up a pretty high performance improvement than NM and AC algorithms.

Fig. 9 presents the experiment results with varied CPU frequencies of MEC nodes which varies from [0.1,0.5] to [0.5,0.9] GHz. We can observe that the average completion time of all tasks for three algorithms is decreased with increased CPU frequencies of MEC nodes. Compared to [0.1,1.0] GHz, performance improvements of COMA-based distributed task migration algorithm for other cases are slightly small. That's because task migration scheme has no obvious improvement on the average completion time, when computing resources are limited or abundant. Nevertheless, no matter how CPU frequencies of MEC nodes varies, multi-agent reinforcement learning algorithm always outperforms NM and AC algorithms, which illustrates the advantage of COMA-based distributed task migration algorithm to minimize the average completion time of tasks.

Next, we will present experimental result for the scalability of COMA algorithm. Critic network and actor network of COMA is trained with 60 users for task migration. And we can use trained actor network to perform task migration for other small numbers of users. As is shown in Fig. 10a, trained actor network with 60 users can be applied to other small umber of users. And with changed number of users, COMA always can achieve a good performance improvement on the average completion time of tasks, comparing with AC and NM algorithms. On the other hand, when number of users is lager than 60, we can process users by dividing users into groups by 60. Then each group is processed in in batches. We also conduct a larger scale experiment with 80 users and 16 MEC nodes. As is shown in Fig. 10b, COMA can achieve up a good performance improvement on the average completion time of tasks with larger scale of task migration problem. In addition, the average completion time of three algorithms increase so much with increased number of users. The reason lies in that number of MEC nodes is unchanged, thus the computation resource becomes more intensive. However, COMA also have a satisfactory performance improvement, compared with NM algorithm and AC algorithm. Thus the COMA task migration algorithm is scalable to for task migration problem. Therefore, the COMA task migration algorithm is scalable to different scale of task migration problem.

On the other hand, we also change migration energy budget $E_{budget}$ from 50 to 450 with other variables remain unchanged. From Fig. 11, we find that the average completion time of all tasks for NM algorithm tends to be unchanged. As we said before, in NM algorithm, tasks are

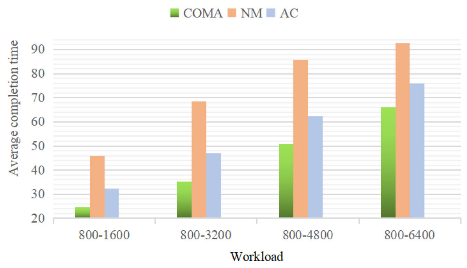
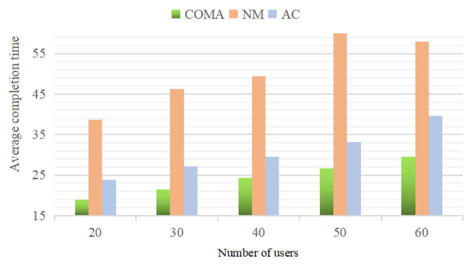
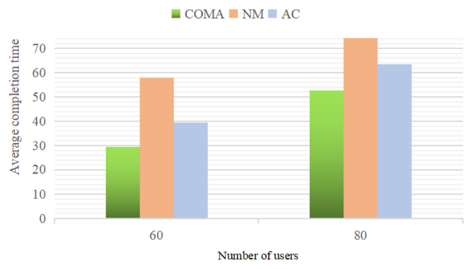


Fig. 8. The average completion time of all tasks with the variation of workload.



(a) smaller scale of task migration



(b) larger scale of task migration

Fig. 10. The average completion time of all tasks with varied number of users.

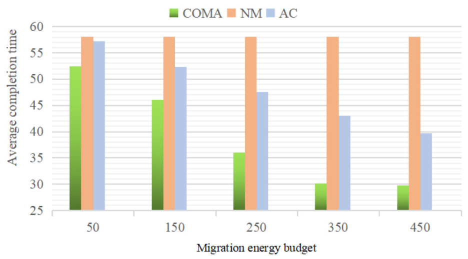


Fig. 11. The average completion time of all tasks with the variation of $E_{budget}$.

executed on MEC node that they are first offloaded to until they are completed, no matter where mobile users that they belongs is moving to. Thus migration energy budget has no effect on NM algorithm. For COMA-based distributed task migration algorithm and AC algorithm, the average completion time of all tasks is decreased with increased migration energy budget. In addition, compared to NM and AC algorithms, performance improvement of COMA-based distributed task migration algorithm grows with increased migration energy budget. To sum up, all experimental results that verify advantages of the proposed COMA-based distributed task migration algorithm.

## 6 Conclusion and Future Works

Migration problem caused by distributed user mobility, which can't be ignored with quality of service (QoS) consideration. In this paper, we investigated task migration problem with multiple users under migration energy budget. Reinforcement learning algorithm based on Markov chain model was applied to solve this problem with low computation complexity, due to the memoryless of user movement. To facilitate cooperation among users, we devised a distributed task migration algorithm based on COMA to minimize the average completion time of tasks under migration energy budget. Extensive experiments were carried out to evaluate the proposed COMA-based distributed task migration algorithm. Compared with no migrating (NM) and single-agent actor-critic (AC) algorithms, the proposed COMA-based distributed task migration algorithm can achieve up 30-50 percent reduction in metrics of average completion time. As part of future directions, we plan to make offloading decision and migration decision simultaneously for tasks with user mobility consideration.

## Acknowledgments

The authors would like to express their gratitude to the anonymous reviewers whose constructive comments have greatly helped to improve this manuscript. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1701403, in part by the Programs of the National Natural Science Foundation of China under Grants 62072165, 61876061, and U19A2058, and in part by Zhijiang Lab, China, under Grant 2020KE0AB01.

## References

[1] T.-L. Chou and L.-J. ChanLin, "Augmented reality smartphone environment orientation application: A case study of the Fu-Jen university mobile campus touring system," *Procedia-Soc. Behav. Sci.*, vol. 46, pp. 410–416, 2012.
[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.
[3] A. Yousefpour *et al.*, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Archit.*, vol. 98, pp. 289–330, 2019.
[4] M. Patel *et al.*, "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-Edge Comput. Ind. Initiative*, pp. 1089–7801, 2014.
[5] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.
[6] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Third Quarter 2017.
[7] A. Nadembega, A. S. Hafid, and R. Brisebois, "Mobility prediction model-based service migration procedure for follow me cloud to support QoS and QoE," in *Proc. IEEE Int. Conf. Commun.*, 2016, pp. 1–6.
[8] A. Nadembega, A. Hafid, and T. Taleb, "A destination and mobility path prediction scheme for mobile networks," *IEEE Trans. Veh. Technol.*, vol. 64, no. 6, pp. 2577–2590, Jun. 2015.
[9] J. Plachy, Z. Becvar, and E. C. Strinati, "Dynamic resource allocation exploiting mobility prediction in mobile edge computing," in *Proc. IEEE 27th Annu. Int. Symp. Pers., Indoor Mobile Radio Commun.*, 2016, pp. 1–6.
[10] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.
[11] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, "On enabling 5G automotive systems using follow ME edge-cloud concept," *IEEE Trans. Veh. Technol.*, vol. 67, no. 6, pp. 5302–5316, Jun. 2018.
[12] Y. Zhai, Y. Wang, I. You, J. Yuan, Y. Ren, and X. Shan, "A DHT and MDP-based mobility management scheme for large-scale mobile internet," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2011, pp. 379–384.
[13] C. Zhang and Z. Zheng, "Task migration for mobile edge computing using deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 96, pp. 111–118, 2019.
[14] Z. Gao, Q. Jiao, K. Xiao, Q. Wang, Z. Mo, and Y. Yang, "Deep reinforcement learning based service migration strategy for edge computing," in *Proc. IEEE Int. Conf. Serv.-Oriented Syst. Eng.*, 2019, pp. 116–1165.
[15] D. Zeng, L. Gu, S. Pan, J. Cai, and S. Guo, "Resource management at the network edge: A deep reinforcement learning approach," *IEEE Netw.*, vol. 33, no. 3, pp. 26–33, May/Jun. 2019.
[16] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 712–725, Sep./Oct. 2019.
[17] Z. Rejiba, X. Masipbruin, and E. Marintordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Comput. Surv.*, vol. 52, no. 5, 2019, Art. no. 90.
[18] Q. Wu, X. Chen, Z. Zhou, and L. Chen, "Mobile social data learning for user-centric location prediction with application in mobile edge service migration," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7737–7747, Oct. 2019.
[19] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.
[20] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Oct. 2018.
[21] T. Taleb and A. Ksentini, "An analytical model for follow me cloud," in *Proc. IEEE Glob. Commun. Conf.*, 2013, pp. 1291–1296.
[22] T. Taleb, A. Ksentini, and P. A. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 369–382, Second Quarter 2019.
[23] S. Wang, R. Urgaonkar, T. He, M. Zafer, K. Chan, and K. K. Leung, "Mobility-induced service migration in mobile micro-clouds," in *Proc. IEEE Military Commun. Conf.*, 2014, pp. 835–840.
[24] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. S. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE ACM Trans. Netw.*, vol. 27, no. 3, pp. 1272–1288, Jun. 2019.

[25] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7457–7469, Aug. 2019.

[26] T. L. Duc, R. G. Leiva, P. Casari, and P. Ostberg, "Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey," *ACM Comput. Surv.*, vol. 52, no. 5, p. 94, 2019.

[27] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[28] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *Siam J. Control Optim.*, vol. 42, no. 4, pp. 1143–1166, 2003.

[29] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. 12th Int. Conf. Neural Inf. Process. Syst.*, vol. 12, pp. 1057–1063, 1999.

[30] M. Tan, "Multi-agent reinforcement learning: Independent versus cooperative agents," *Readings in Agents.* San Francisco, CA, USA: Morgan Kaufmann, 1997, pp. 487–494.

[31] P. Hernandezleal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Auton. Agents Multi-Agent Syst.*, vol. 33, no. 6, pp. 750–797, 2019.

[32] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. 22nd AAAI Conf. Artif. Intell.*, 2018, pp. 2974–2982.

**Chubo Liu** (Member, IEEE) received the BS and PhD degrees in computer science and technology from Hunan University, China, in 2011 and 2016, respectively. He is currently an associate professor of Computer Science and Technology with Hunan University. His research interests includes game theory, approximation and randomized algorithms, cloud, and edge computing. He has published more than 30 papers in journals and conferences such as the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Industrial Informatics*, *IEEE Internet of Things Journal*, *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, *Theoretical Computer Science*, ICPADS, HPCC, and NPC. He won the Best Paper Award in IFIP NPC 2019 and the IEEE TCSC Early Career Researcher(ECR) Award, in 2019. He is a member of CCF and IEEE.

**Fan Tang** received the BS degree in computer science and technology from Huaqiao University, China, in 2018. She is currently working toward the MS degree at Hunan University, China. Her research interests include cloud computing, edge computing, reinforcement learning, and game theory.

**Yikun Hu** received the PhD degree from Hunan University, China. His research interests includes parallel and distributed processing, cluster, grid and cloud computing.

**Kenli Li** (Senior Member, IEEE) received the PhD degree in computer science from Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar at the University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently the dean and a full professor of computer science and technology with Hunan University, and deputy director of National Supercomputing Center in Changsha. His major research interests include parallel computing, high-performance computing, and grid and cloud computing. He has published more than 230 research papers in international conferences and journals such as the *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Signal Processing*, *Journal of Parallel and Distributed Computing*, ICPP, and CCGrid. He is an outstanding member of CCF. He serves on the editorial board of *IEEE Transactions on Computers*.

**Zhuo Tang** received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2008. He is currently a full professor with the College of Computer Science and Electronic Engineering, Hunan University, and is the associate chair of the department of computing science. His majors are distributed computing system, cloud computing, and parallel processing for big data, including distributed machine learning, security model, parallel algorithms, and resources scheduling and management in these areas. He is a member of ACM and CCF.

**Keqin Li** (Fellow, IEEE) is a SUNY distinguished professor of computer science. His current research interests include parallel computing and high performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of Things, and cyber-physical systems. He has published more than 630 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently serving or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*.