Energy-Efficient Dependency-Aware Task Offloading in Mobile Edge Computing: A Digital Twin Empowered Approach

Huan Zhou*, Lingxiao Chen*, Kai Jiang†, and Yuan Wu‡
*College of Computer and Information Technology, China Three Gorges University, Yichang, China
†School of Cyber Science and Engineering, Wuhan University, Wuhan, China
‡ State Key Laboratory of Internet of Things for Smart City, University of Macau, Macau, China

2022 IEEE 10th International Conference on Smart City and Informatization (iSCI) | 979-8-3503-9631-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISCI57775.2022.00018

***Abstract*—In the 6G era, integration of Digital Twin (DT) and mobile edge computing (MEC) has been expected to significantly improve the service quality of many mobile applications. However, existing studies rarely consider service caching and task dependency together, resulting in a degraded system performance. In addition, the collaboration between Edge Servers (ESs) should also be taken into account because of their limited computing resources as well as caching capacities. In this paper, we investigate a DT-empowered MEC architecture, which intelligently offloads tasks of Mobile Users (MUs) to collaborative ESs with the assistance of DT, while accounting for the service caching and task dependency. Accordingly, we formulate the problem as a Mixed Integer Non-linear Programming (MINLP) problem, aiming to minimize the system-wise energy consumption. In order to solve this problem, the Asynchronous Advantage Actor-Critic (A3C)-based algorithm is proposed. Extensive simulation results demonstrate that our proposed algorithm can reduce the long-term energy consumption of the system greatly, and outperforms the other benchmark algorithms under different scenarios.**

***Index Terms*—Mobile Edge Computing, Digital Twin, Service Caching, Dependency.**

I. Introduction

With the rapid development of B5G/6G technologies and the wide use of intelligent terminals, computation-intensive and delay-sensitive mobile applications such as augmented reality, face recognition and virtual reality, have also developed rapidly. In order to promote the Quality of Service (QoS) of Mobile Users (MUs), Mobile Edge Computing (MEC) has recently emerged to support high-performance computing, especially for delay-sensitive yet computation-intensive applications by sinking computing power to the network edge [1]–[5].

Digital Twin (DT) is a powerful emerging technology that enables the mapping of real-world devices in the virtual space [6], [7]. Nowadays, artificial intelligence (AI), especially Deep Reinforcement Learning (DRL), is widely used in MEC scenarios [8]. Due to the limitations of computing power and storage space, MUs cannot store a large amount of data, and do not have enough computing power to train neural networks. To address this issue, integration of MEC and DT provides a promising solution [9]. Specifically, DT is responsible for collecting the data of various physical entities and saving a large amount of data, which can assist in training the neural network for obtaining the optimal strategy.

The existing works on MEC mainly focus on computation offloading, while rarely accounting for the service caching and task dependency [10]–[12]. Nevertheless, both the service caching and task dependency play crucial roles in influencing the performance and feasibility of task offloading. First, service caching refers to storing applications and data on edge servers (ESs) which execute specific computing tasks and thus reduce the delay of task completion [13]–[15]. Second, many computation-intensive tasks are comprised of multiple dependent subtasks, and the current subtask can be processed only after the previous subtask is completed [16]–[18]. For those dependent tasks, making a series of optimal offloading decisions is more than challenging, since the offloading decisions of the current subtask will affect the decisions of subsequent subtasks. To the best of the authors' knowledge, there exist few research efforts investigating the service caching and task dependency at the same time. In [19], the authors proposed a cache-assisted single-user and single-edge server system, where the server can selectively cache the previously generated programs for future reuse. In [20], the authors studied how to efficiently offload dependent tasks to edge nodes with limited caching resources, aiming to find a feasible offloading solution with a minimum makespan.

Nevertheless, the aforementioned studies do not consider the allocation of computing resources, and moreover, the application scenario is relatively simple, e.g., without considering the collaboration among the ESs. In this paper, we take into account the task dependency, service caching, as well as edge collaboration in multi-user and multi-server MEC system. Focusing on this system, we investigate the corresponding joint optimization of computation offloading and resource allocation, with the objective of minimizing the system-wise energy consumption. In addition, DT is utilized to realize the intelligent task offloading, and an Asynchronous Advantage Actor-Critic (A3C)-based algorithm is proposed for determining the optimal offloading and resource allocation strategy that can minimize the system-wise energy consumption. The main contributions of this paper are summarized as follows:

979-8-3503-9631-7/22/$31.00 ©2022 IEEE
DOI 10.1109/iSCI57775.2022.00018

1) A DT-empowered MEC system architecture is proposed, in which DT is used to collect a large amount of data and train the neural network to make the prediction more accurate.
2) Considering service caching and task dependency, the joint optimization of computation offloading and resource allocation is described as a Mixed Integer Non-Linear Programming (MINLP) problem to minimize long-term system energy consumption. Then, the A3C-based algorithm is proposed to solve this problem.
3) The simulation results show that the proposed A3C-based algorithm achieve a better performance than other benchmark algorithms in different scenarios.

The remainder of this paper is organized as follows. Section II introduces the system model and problem formulation. Section III proposes the A3C-based algorithm. Section IV provides the simulation results. Finally, conclusions are summarized in Section V.

II. SYSTEM MODEL

In this section, we present the system model adopted in this paper, including the network architecture, communication model, computation offloading model, and the corresponding problem formulation.

A. Network Architecture

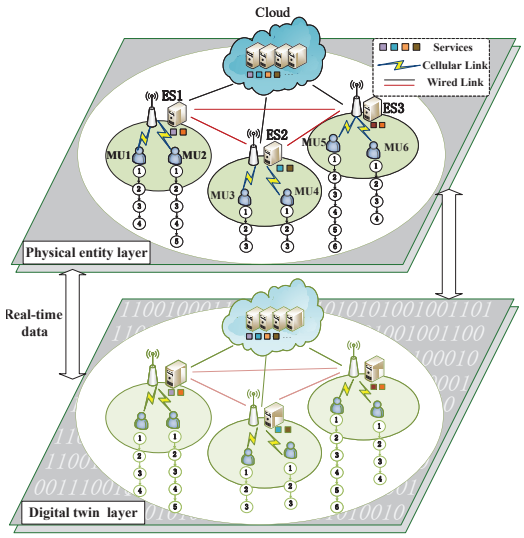

Fig. 1. The Architecture of Digital Twin Network.

As shown in Fig. 1, we consider a DT-empowered MEC architecture with K ESs and a cloud server, which can be divided into the physical entity layer and the DT layer. The set of these ESs is denoted by $\mathcal{K} = \{1, 2, \ldots, K\}$. In the physical entity layer, U MUs (denoted by $\mathcal{U} = \{1, 2, \ldots, U\}$) are randomly scattered in the wireless coverage scope and can offload tasks to ESs since their limited computing and storage resources. In the DT layer, terminal devices and edge devices are equipped with corresponding sensors, which can mirror and map the relevant hardware configuration, historical data, network status and other information to the DT layer. Furthermore, DT constantly interacts with the physical system to make efficient decisions in real-time.

Each MU $u \in \mathcal{U}$ will generate a computation-intensive task such as face recognition, online games, virtual reality, etc., which can be divided into multiple interdependent subtasks, denoted by $\mathcal{I} = \{1, 2, \ldots, I\}$. In addition, subtasks are interdependent, which means that the input of the latter subtask $i (i \in \mathcal{I})$ is the output of the previous subtask $i'(i' \in \mathcal{I})$. Thus, only after the previous subtask i' is completed, the next subtask can start calculation. Next, the parameters of the subtask information are described in detail. $c_{u,i}$ denotes the total number of CPU cycles required to complete the subtask, and $d_{u,i}$ indicates the data size of the subtask. Further, each subtask has its maximum tolerance delay expressed by $t_{u,i}^{max}$.

Besides, each ES $k (k \in \mathcal{K})$ is endowed with a limited available caching capacity, such that only a small portion of services can be cached in it. Accordingly, we define $\mathcal{Q}$ to indicate the set of subtask types, $\mathcal{Q} = \{1, 2, \ldots, Q\}$, and each subtask type $q \in \mathcal{Q}$ requires a storage space h_q to cache the corresponding services. Then, let $\varphi_{k,q} = \{0, 1\}$ denote whether ES k caches the service related to subtask type q, with $\varphi_{k,q} = 1$ denoting that the service pertaining to subtask type q is cached at ES k. Generally speaking, the cloud is able to cache all services as its abundant caching capacity. Moreover, it is assumed that the ESs in a cluster form can cooperatively provide distributed caching to make full use of the available caching resources.

B. Communication Model

The communication between two different servers (including both the one between the ES and the cloud server, and the one between two different ESs) is carried out through optical fiber wired connection. We assume that r_{es} is the transmission rate between two different ESs, and r_i^{cloud} represents the transmission rate between each ES and the cloud server. Hence, the transmission delay and energy consumption between ESs are, respectively, given by

$$t_{u,k,i}^{trs} = \frac{d_{u,i}}{r_{es}}, \tag{1}$$

$$e_{u,k,i}^{trs} = P_k^{trs} d_{u,i}, \tag{2}$$

where $P_k^{trs}(J/bit)$ indicates the energy consumption per bit of data transmission between different ESs. Then, the transmission delay and energy consumption between ESs and the cloud are, respectively, given by

$$t_{u,k,i}^{cloud} = \frac{d_{u,i}}{r_i^{cloud}}, \tag{3}$$

$$e_{u,k,i}^{cloud} = P_{cloud} d_{u,i}, \tag{4}$$

where $P_{cloud}(J/bit)$ indicates the energy consumption per bit of data transmission between the ES and the cloud server.

C. Computation offloading Model

1) **ES Computation Offloading**: The computing resources allocated by ES k to the subtask i is denoted as $f_{u,k,i}$. When the subtask i is executed on ES k, the corresponding computing delay is given by

$$t_{u,k,i}^{comp} = \frac{c_{u,i}}{f_{u,k,i}}. \tag{5}$$

Then, the corresponding energy consumption executed on ES k can be expressed as

$$e_{u,k,i}^{comp} = P_k^{comp} t_{u,k,i}^{comp}, \tag{6}$$

where P_k^{comp} indicates the computation power.

2) **Cloud Computation Offloading**: In this case, if the ES is unable to complete computing tasks due to limited computing resources or caching capacity, its subtasks can be offloaded to the cloud server. Considering the abundant computing resources in the cloud server, the computing delay and energy consumption on the cloud server can be ignored. Only the transmission delay $t_{u,k,i}^{cloud}$ and energy consumption $e_{u,k,i}^{cloud}$ offloading subtasks to the cloud are considered.

For each subtask, we can choose to offload to the ES or the cloud for execution. Therefore, we define the related binary offloading decision variables. $a_{u,k,i} = \{0,1\}$ is defined to denote whether the subtask i is executed on ES k. For the sake of illustration, $a_{u,k',i'} = 1$ denotes that the previous subtask i' is executed on ES k'. In addition, we define $a_{u,c,i} = \{0,1\}$ to denote whether the subtask i is executed on the cloud. Same as above, $a_{u,c,i'} = \{0,1\}$ denotes whether the previous subtask i' is executed on the cloud.

For the subtask i of MU u, the following situations are considered according to the place where the subtask is performed and the input data source of the subtask.

a) The subtask is executed in the ES, and the input of this subtask comes from the same ES $(a_{u,k',i'} = 1, a_{u,k,i} = 1, k' = k)$. In this case, two consecutive subtasks are executed in an ES, which will not cause additional task data transmission delay and energy consumption.

b) The subtask is executed in the ES, and the input of this subtask comes from other ES $(a_{u,k',i'} = 1, a_{u,k,i} = 1, k' \neq k)$. In this case, two consecutive subtasks are executed in two different ESs, which will cause additional task data transmission delay and energy consumption.

c) The subtask is executed in the ES, and the input of this subtask comes from the cloud $(a_{u,c,i'} = 1, a_{u,k,i} = 1)$. In this case, two consecutive subtasks are executed in the cloud and ES, respectively, which will cause additional task data transmission delay and energy consumption.

d) The subtask is executed in the cloud, and the previous related subtask i' is also executed in the cloud $(a_{u,c,i'} = 1, a_{u,c,i} = 1)$. In this case, two consecutive subtasks are executed in the cloud, which will not cause additional task data transmission delay and energy consumption.

e) The subtask is executed in the cloud, and the input of this subtask comes from the ES $(a_{u,c,i'} = 1, a_{u,k,i} = 1)$. Similar to case c, it will cause additional task data transmission delay and energy consumption.

Based on the above, the completion time of subtask i generated by MU u can be expressed as

$$t_{u,i}^{fin} = \begin{cases} t_{u,i'}^{fin} + t_{u,k,i}^{comp}, & a_{u,k',i'} = 1, a_{u,k,i} = 1, k' = k, \\ t_{u,i'}^{fin} + t_{u,k,i}^{trs} + t_{u,k,i}^{comp}, & a_{u,k',i'} = 1, a_{u,k,i} = 1, k' \neq k, \\ t_{u,i'}^{fin} + t_{u,k,i}^{cloud} + t_{u,k,i}^{comp}, & a_{u,c,i'} = 1, a_{u,k,i} = 1, \\ t_{u,i'}^{fin}, & a_{u,c,i'} = 1, a_{u,c,i} = 1, \\ t_{u,i'}^{fin} + t_{u,k,i}^{cloud}, & a_{u,k,i'} = 1, a_{u,c,i} = 1, \end{cases} \tag{7}$$

where $t_{u,i'}^{fin}$ denotes the completion time of subtask i' generated by MU u. Therefore, the maximum completion delay of correlation tasks generated by MU u can be expressed as $T_u = \max\left\{t_{u,i}^{fin}, i \in I\right\}$. In addition, the energy consumption generated by completing subtask i is

$$e_{u,i}^{fin} = \begin{cases} e_{u,i'}^{fin} + e_{u,k,i}^{comp}, & a_{u,k',i'} = 1, a_{u,k,i} = 1, k' = k, \\ e_{u,i'}^{fin} + e_{u,k,i}^{trs} + e_{u,k,i}^{comp}, & a_{u,k',i'} = 1, a_{u,k,i} = 1, k' \neq k, \\ e_{u,i'}^{fin} + e_{u,k,i}^{cloud} + e_{u,k,i}^{comp}, & a_{u,c,i'} = 1, a_{u,k,i} = 1, \\ e_{u,i'}^{fin}, & a_{u,c,i'} = 1, a_{u,c,i} = 1, \\ e_{u,i'}^{fin} + e_{u,k,i}^{cloud}, & a_{u,k,i'} = 1, a_{u,c,i} = 1, \end{cases} \tag{8}$$

Similarly, the system energy consumption corresponding to the task completed by a MU can be expressed as $E_u = \max\left\{e_{u,i}^{fin}, i \in I\right\}$.

D. Problem Formulation

Our goal is to minimize the total energy consumption of all MUs in the system by optimizing the offloading decision and computing resource allocation decision. Therefore, this problem can be expressed as follows

$$\min_{\mathbf{a},\mathbf{f}} \quad \sum_{u=1}^{U} E_u \tag{9}$$

$$s.t. \quad a_{u,k,i} = \{0,1\}, \forall u \in \mathcal{U}, \forall k \in \mathcal{K}, \forall i \in \mathcal{I} \tag{9a}$$

$$a_{u,c,i} = \{0,1\}, \forall u \in \mathcal{U}, \forall i \in \mathcal{I} \tag{9b}$$

$$\sum_{i \in I} a_{u,k,i} + a_{u,c,i} = 1, \forall u \in \mathcal{U}, \forall k \in \mathcal{K} \tag{9c}$$

$$\sum_{q \in Q} \varphi_{k,q} h_q \leq C_k, \forall k \in \mathcal{K} \tag{9d}$$

$$\sum_{i \in I} f_{u,k,i} \leq f_k^{edge}, \forall k \in \mathcal{K} \tag{9e}$$

$$t_{u,i}^{fin} - t_{u,i'}^{fin} \leq t_{u,i}^{max}, \forall u \in \mathcal{U}, \forall i \in \mathcal{I}, \forall i' \in \mathcal{I} \tag{9f}$$

$$T_u \leq T_u^{max}, \forall u \in \mathcal{U}, \tag{9g}$$

where (9c) indicates that each subtask is either executed on ES k or on the cloud; (9d) indicates that the total size of cached services should not exceed the storage capacity of ES k, where C_k indicates the maximum caching capacity of ES k; (9e) ensures that the computing resources allocated to MUs are not greater than the available computing resources f_k^{edge} provided by ES k; (9f) and (9g) respectively represent the limit of delay for each subtask and the entire subtasks. Especially, T_u^{max} is the maximum tolerance delay executed the entire subtasks.

III. PROPOSED A3C BASED ALGORITHM

In this section, in order to minimize the energy consumption under latency and services type constraint, we propose an A3C-based algorithm to solve the optimization problem. We first illustrate the definitions of the state space, action space and reward.

A. Definition of the State, Action and Reward

Three critical elements are identified in the DRL based method, i.e., state, action and reward, which are defined as:

1) **State space (S)**: The state in MDP is used to reflect the state space of the network environment. Therefore, the state at time slot t can be expressed as $s(t) = \{\mathcal{E}(t), X(t)\}$, where $\mathcal{E}(t)$ indicates the state of ESs, including the set of services caching, available computing resources and caching capacities of ES. $X(t) = \{x^t_{1,1,i}, x^t_{1,2,i}, \ldots, x^t_{1,k,i}, x^t_{2,1,i}, x^t_{2,2,i}, \ldots, x^t_{2,k,i}, x^t_{u,1,i}, x^t_{u,2,i}, \ldots, x^t_{u,k,i}\}$ indicates the input data source of the current subtask.

2) **Action space (A)**: Action space is utilized to represent computation offloading decisions and resource allocation decisions at each time slot. The offloading decision vector can be indicated as $X(t)$, and the computing resource allocation can be indicated as $F(t) = \left\{ f^t_{u,k,i} \mid u \in \mathcal{U}, k \in \mathcal{K}, i \in \mathcal{I} \right\}$.

3) **Reward (R)**: The agent will receive a immediate reward according to current state S and action A. To minimize system energy consumption, the reward function can be defined as: $R(s(t), a(t)) = -\sum_{u=1}^{U} e^{fin}_{u,i} - e^{fin}_{u,i'}(t)$.

B. A3C-based Algorithm for Determining the Computation Offloading and Resource Allocation

In the A3C algorithm, there is a global neural network model that stores the network parameters. After each thread interacts with the environment to a certain amount of data, the global neural network is updated according to the gradient of the neural network loss function in its own thread. Then, workers will update their neural network parameters to global neural network parameters at set intervals, to guide the subsequent environmental interactions.

Next, the workflow of A3C will be explained in detail. Initially, DT collects information from the environment to form a state space. After obtaining the current state s_t, it takes a corresponding action according to the policy function $\pi(a_t \mid s_t; \theta)$ and then moves to the next state s_{t+1}, and will obtain an immediate reward r_t. The state value function $V(s_t; \theta_v)$ with parameter θ_v can be expressed as

$$\begin{aligned} V(s_t; \theta_v) &= E[G_t \mid s = s_t, \pi] \\ &= E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s = s_t, \pi\right], \end{aligned} \tag{10}$$

where G_t indicates the discounted accumulated reward under state s_t. $\gamma \in [0, 1]$, is the discount factor, representing the impact of future returns on the current status value.

In A3C, the k-step update method is adopted for parameter updating instead of one-step return, to accelerate the learning process. Hence, the cumulative reward of k-step can be given by

$$R_t = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v), \tag{11}$$

where r_{t+i} is the immediate reward. k is upper-bounded by t_{max} which varies with different states, and both the policy and value functions are updated after t_{max} action is taken or when the final state is reached.

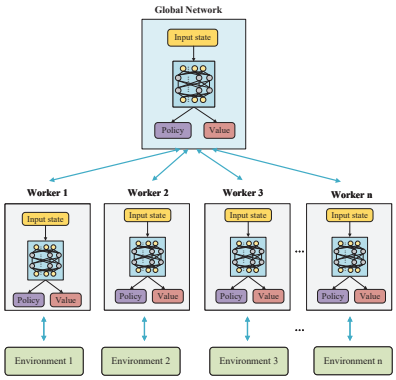

Fig. 2. Framework of A3C algorithm.

In addition, the advantage function is defined to significantly reduce the variance in gradient calculation, which can be expressed as

$$\begin{aligned} A(s_t, a_t; \theta, \theta_v) &= R_t - V(s_t; \theta_v) \\ &= \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v), \end{aligned} \tag{12}$$

where θ and θ_v are the parameters of the policy (the actor) $\pi(a_t \mid s_t; \theta)$ and the value function (the critic). The advantage function can be employd to evaluate the quality of an action. When the advantage function $A(s_t, a_t)$ is positive, it means that the action is superior than the average action. On the contrary, when the advantage function $A(s_t, a_t)$ is negative, it means that the current action is inferior to the average action.

According to the advantage function, the loss function of the actor can be defined as

$$f_\pi(\theta) = \log \pi(a_t \mid s_t; \theta)(R_t - V(s_t; \theta_v)) + \beta H(\pi(s_t; \theta)), \tag{13}$$

In order to prevent premature convergence to suboptimal strategy due to insufficient training exploration, $H(\pi(s_t; \theta))$ is put into the formula as the entropy term of strategy π. β is a parameter used to control the balance between exploration and exploitation, and higher β prefers to exploration. The loss function of the value function can be expressed as

$$f_v(\theta_v) = (R_t - V(s_t; \theta_v))^2. \tag{14}$$

Then, the iterative formulas of actor and critic are given as

$$\begin{aligned} d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_t \mid s_t; \theta')(R_t - V(s_t; \theta_v)) \\ + \delta \nabla_{\theta'} H(\pi(s_t; \theta')). \end{aligned} \tag{15}$$

Algorithm 1 A3C-based joint Computation Offloading and Resource Allocation algorithm.

Input: The parameters about the testbed environments;

Output: The optimal computation offloading strategy and resource allocation strategy;

Initialization:

Initialize the global actor and global critic network parameters: θ and θ_v;

Initialize the thread-specific actor parameter θ_v and and thread- specific critic network parameter θ'_v;

Initialize global shared counter $T \leftarrow 0$ and thread step counter $t \leftarrow 1$;

Initialize the parameters of simulation environment setup;

Iteration:

repeat

  for each worker **do**

    Set gradients of two global networks: $d\theta = 0$, $d\theta_v = 0$;

    Synchronous parameters of each worker with global parameters: θ and θ_v;

    Obtain the current state s_{t+1};

    for $t < t_{\max}$ **do**

      Perform action according to policy $\pi\left(a_t \mid s_t; \theta\right)$;

      Receive reward r_t and new state s_{t+1};

      $t \leftarrow t + 1$;

    end for

    $$R = \begin{cases} 0, & \text{for terminal state} \\ V\left(s_t, \theta'_v\right), & \text{for non - terminal state.} \end{cases}$$

    for $i \in \{t-1, \cdots, t_{\text{start}}\}$ **do**

      Update $R = r_t + \gamma R$;

      Obtain cumulative gradient wrt θ' by Eq. (15);

      Obtain cumulative gradient wrt θ'_v by Eq. (16);

    end for

    Asynchronous update θ and θ_v;

    $T \leftarrow T + 1$;

  end for

until $T > T_{\max}$

and

$$d\theta_v \leftarrow d\theta_v + \frac{\partial \left(R_t - V\left(s_t; \theta_v\right)\right)^2}{\partial \theta'_v}. \tag{16}$$

The global parameter θ and θ_v update asynchronously based on the RMSProp algorithm.

The details of the proposed A3C-based algorithm can be summarized by Algorithm 1.

IV. NUMERICAL RESULTS

In order to evaluate the performance of our proposed algorithm, we consider a dynamic MEC scenario, including several ESs, multiple MUs, and a cloud. It is assumed that there are 3 ESs and 6 MUs in our scenario. For computing subtasks generated by MUs, the number of subtask types is set to 10, and the data size of each subtask is between [1, 4] Mbit. For each ES, the computing resources and caching capacity are set to [4, 8] GHz and [10, 60] GB, respectively, and each service storage requirement is [1, 4] GB. In addition, the energy consumption per bit of data transmission between ESs $P_k^{trs} = 3 \times 10^{-8} (J/bit)$, and the energy consumption per bit of data transmission between the ES and the cloud server $P_{cloud}(j/bit) = 2 \times 10^{-7} (J/bit)$.

In addition, for a more comprehensive evaluation of our proposed A3C-based algorithm, it is compared with DDPG algorithm and Greedy algorithm as well as other benchmark schemes as follows. Cloud Execution (CE) scheme considers that all subtasks are executed in the cloud. Random Offloading (RO) scheme considers that each subtask is offloaded randomly to the ES that caches the corresponding service. If no ES caches this service, it has to be offloaded to the cloud.

In Fig. 3, we compare the convergence performance of the proposed A3C-based algorithm and DDPG algorithm. It can be found that the proposed A3C-based algorithm is slightly better than DDPG in the final convergence results. At the same time, the convergence speed of the proposed A3C-based algorithm is much faster than that of DDPG. This is because the proposed A3C-based algorithm utilizes multi-threading to interact with the environment, which improves the training effect.

Fig. 4 shows the relationship between the energy consumption and the number of MUs. It can be observed that the proposed A3C-based algorithm performs the best, followed by DDPG. The reason is that they consider computation offloading and resource allocation on the basis of service caching and edge collaboration. Greedy algorithm usually obtains the local optimal solution rather than the global optimal solution. As a result, the performance of Greedy algorithm is worse than that of the proposed A3C-based algorithm. As the number of MUs increases, the energy consumption of the five methods increases correspondingly. This is because as the number of MUs increases, more computing tasks will be generated, and the competition for computing resources will become more intense.

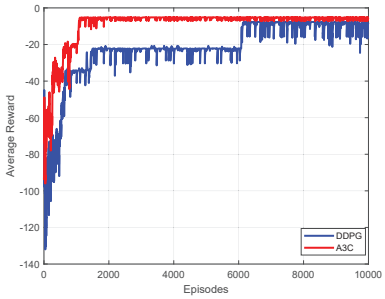

Fig. 3. The convergence of the proposed A3C-based algorithm and DDPG algorithm.

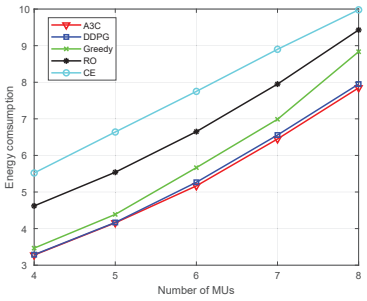

Fig. 4. Energy consumption versus the number of MUs.

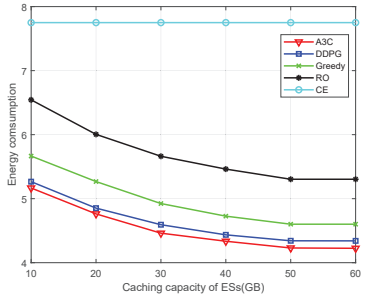

Fig. 5. Energy consumption versus the caching capacity of ESs

Fig. 5 shows the relationship between the energy consumption and the caching capacity of ESs. It can be observed that as the caching capacity of ESs increases, the energy consumption of all schemes will decrease except CE. This is because the ES can cache more services, and the number of subtasks that need to be offloaded to other ES or the cloud will decrease, resulting in a reduction in transmission energy consumption. For CE, no caching resources of the ESs are used, so the energy consumption value remains unchanged.

In summary, the proposed A3C-based algorithm outperforms other benchmark methods in terms of the long-term energy consumption of the system under different scenarios, which validates the effectiveness of our proposed method.

V. CONCLUSION

In this paper, we have proposed a DT-empowered MEC system architecture, considering both the service caching and task dependency simultaneously. First, the optimization problem is formalized as a MINLP problem, and then the A3C-based algorithm is proposed to minimize the long-term energy consumption of the system and proivde the optimal strategy of computation offloading and resource allocation. The simulation results show that the proposed algorithm can reduce the long-term energy consumption of the system and outperform the benchmark schemes under different scenarios.

ACKNOWLEDGEMENT

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants NO. 62172255 and NO. 61872221, and the Science and Technology Development Fund of Macau SAR under Grants 0060/2019/A1 and 0162/2019/A3.

REFERENCES

[1] X. Shang, Y. Huang, Y. Mao, Z. Liu, and Y. Yang, "Enabling qoe support for interactive applications over mobile edge with high user mobility," in *Proc. IEEE INFOCOM*, 2022, pp. 1289–1298.
[2] H. Zhou, T. Wu, H. Zhang, and J. Wu, "Incentive-driven deep reinforcement learning for content caching and d2d offloading," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2445–2460, 2021.
[3] Y. Qu, H. Dai, H. Wang, C. Dong, F. Wu, S. Guo, and Q. Wu, "Service provisioning for uav-enabled mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 11, pp. 3287–3305, 2021.
[4] H. Zhou, X. Chen, S. He, J. Chen, and J. Wu, "Draim: A novel delay-constraint and reverse auction-based incentive mechanism for wifi offloading," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 4, pp. 711–722, 2020.
[5] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet of Things J.*, vol. 9, no. 2, pp. 1517–1530, 2021.
[6] T. Liu, L. Tang, W. Wang, Q. Chen, and X. Zeng, "Digital-twin-assisted task offloading based on edge collaboration in the digital twin edge network," *IEEE Internet of Things J.*, vol. 9, no. 2, pp. 1427–1444, 2021.
[7] X. Lin, J. Wu, J. Li, W. Yang, and M. Guizani, "Stochastic digital-twin service demand with edge response: An incentive-based congestion control approach," *IEEE. Trans. Mob. Comput.*, 2021.
[8] K. Jiang, C. Sun, H. Zhou, X. Li, M. Dong, and V. C. Leung, "Intelligence-empowered mobile edge computing: Framework, issues, implementation, and outlook," *IEEE Netw.*, vol. 35, no. 5, pp. 74–82, 2021.
[9] B. Fan, Y. Wu, Z. He, Y. Chen, T. Q. Quek, and C.-Z. Xu, "Digital twin empowered mobile edge computing for intelligent vehicular lane-changing," *IEEE Netw.*, vol. 35, no. 6, pp. 194–201, 2021.
[10] G. Guo and J. Zhang, "Energy-efficient incremental offloading of neural network computations in mobile edge computing," in *Proc. IEEE GLOBECOM*, 2020, pp. 1–6.
[11] Y. Dai, D. Xu, S. Maharjan, G. Qiao, and Y. Zhang, "Artificial intelligence empowered edge computing and caching for internet of vehicles," *IEEE Wirel. Commun.*, vol. 26, no. 3, pp. 12–18, 2019.
[12] Z. Song, Y. Liu, and X. Sun, "Joint task offloading and resource allocation for noma-enabled multi-access mobile edge computing," *IEEE Trans. Commun.*, vol. 69, no. 3, pp. 1548–1564, 2020.
[13] Z. Xu, L. Zhou, H. Dai, W. Liang, W. Zhou, P. Zhou, W. Xu, and G. Wu, "Energy-aware collaborative service caching in a 5g-enabled mec with uncertain payoffs," *IEEE Trans. Commun.*, vol. 70, no. 2, pp. 1058–1071, 2021.
[14] J. Yan, S. Bi, L. Duan, and Y.-J. A. Zhang, "Pricing-driven service caching and task offloading in mobile edge computing," *IEEE Wirel. Commun.*, vol. 20, no. 7, pp. 4495–4512, 2021.
[15] G. Zheng, C. Xu, H. Long, and X. Zhao, "Mec in noma-hetnets: A joint task offloading and resource allocation approach," in *Proc. IEEE WCNC*, 2021, pp. 1–6.
[16] H. Liao, X. Li, D. Guo, W. Kang, and J. Li, "Dependency-aware application assigning and scheduling in edge computing," *IEEE Internet of Things J.*, vol. 9, no. 6, pp. 4451–4463, 2021.
[17] M. Mehrabi, S. Shen, V. Latzko, Y. Wang, and F. H. Fitzek, "Energy-aware cooperative offloading framework for inter-dependent and delay-sensitive tasks," in *Proc. IEEE GLOBECOM*, 2020, pp. 1–6.
[18] W. He, L. Gao, and J. Luo, "A multi-layer offloading framework for dependency-aware tasks in mec," in *Proc. IEEE ICC*, 2021, pp. 1–6.
[19] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Trans. Commun.*, vol. 19, no. 7, pp. 4947–4963, 2020.
[20] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *Proc. IEEE INFOCOM*, 2020, pp. 1997–2006.