Deep Reinforcement Learning in Cache-Aided MEC Networks

Zhong Yang, Yuanwei Liu, Yue Chen and Gareth Tyson
School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK

***Abstract*—A novel resource allocation scheme for cache-aided mobile-edge computing (MEC) is proposed, to efficiently offer communication, storage and computing service for intensive computation and sensitive latency computational tasks. In this paper, the considered resource allocation problem is formulated as a mixed integer non-linear program (MINLP) that involves a joint optimization of tasks offloading decision, cache allocation, computation allocation, and dynamic power distribution. To tackle this non-trivial problem, Markov decision process (MDP) is invoked for mobile users and the access point (AP) to learn the optimal offloading and resource allocation policy from historical experience and automatically improve allocation efficiency. In particular, to break the curse of high dimensionality in the state space of MDP, a deep reinforcement learning (DRL) algorithm is proposed to solve this optimization problem with low complexity. Moreover, extensive simulations demonstrate that the proposed algorithm is capable of achieving a quasi-optimal performance under various system setups, and significantly outperform the other representative benchmark methods considered. The effectiveness of the proposed algorithm is confirmed from the comparison with the results of the optimal solution.**

I. Introduction

As the radical growth of smart devices and swift development of many new applications, mobile applications have been growing exponentially in the 5G networks. The prosperous of heterogeneous services and related applications, such as augmented reality (AR), virtual reality (VR), real-time online gaming and high-speed video streaming in the 5G networks require unprecedented high access speed and low latency. A wide range of new applications and services in wireless networks brings tough challenges including a massive number of mobile smart devices, intensive computation and sensitive latency multimedia, applications transmissions and low power consumption. However, the battery capacity and computation resources of mobile devices are limited. If all the computational tasks are executed locally in mobile devices, the performance and delay requirements of the intensive computation and sensitive latency applications are significantly strained by limited battery capacity and computation resources of the mobile devices. To address this demanding challenge, a new trend is appearing with the function of central networks being increasingly moved towards the network edges (see [1–3], and references therein).

In a mobile-edge computing (MEC) system, computing capabilities are provided within the edge of wireless networks in closed proximity to mobile users. The computation tasks are capable of being offloaded to the AP or the base stations (BSs), which liberates the mobile devices from heavy computation workloads and reduces the energy consumption of the system. The key idea of MEC is to promote cloud-computing resources at the edge of mobile networks by integrating MEC servers at the wireless APs or BSs. The computing/storage resources, utilized in both mobile communication and cloud computing services, are presented as virtual resources. A MEC system involves the placement of virtual services at the edge of the networks. However, determining communication and computation resources at the networks edge can cause ferocious challenges for APs or BSs, e.g. placement of resources and managing backhaul to satisfy large numbers of intensive computation and sensitive latency multimedia transmissions and applications.

Meanwhile, in some scenarios, the task computation results maybe requested by other users. For instance, in real-time online gaming, game tiles would be sent to a bunch of nearby users in the same game, thus caching reusable task computation results closer to users is capable of vastly reducing the computation burden and latency [4–7]. In a multi-users MEC system, where there are a bunch of users requesting for services from one AP, the major challenge is computing model selection (i.e., local computing or offloading computing) and computation resources distribution to achieve minimum energy consumption. In this paper, we propose a novel task offloading and resource allocation scheme for cache-aided mobile edge computing, in which computation and cache resources are integrated in the AP for users with intensive computation and sensitive latency tasks.

Although numerous research contributions have been made in the field of MEC scenarios, most of the existing works adopt stochastic geometry for task offloading [8, 9]. Due to the combinatorial nature of computing mode selection, the task offloading problem is generally formulated as mixed integer programming (MIP). To tackle the MIP problem, branch-and-bound algorithms [10] and dynamic programming [11] are used to solve for the globally optimal offloading solution. Compared to before mentioned traditional optimization methodologies, machine learning approached are growing rapidly [12]. In [13, 14], reinforcement learning, more specifically Q-learning, is invoked for an IoT device to choose the MEC device and determine the offloading rate to achieve the optimal offloading performance. Authors in [15] proposed a deep reinforcement learning-based online offloading framework that implements a deep neural network to generate offloading decisions, which, compares to Q-learning, does not require any manually labeled training data. While the aforementioned research contributions have laid a solid foundation on MEC, the investigations on the applications of machine learning in MEC are still in fancy. It is worth pointing out that the characteristics of MEC make it challenging to apply the reinforcement learning, because the number of states increase exponentially with the number of users and tasks. With the

978-1-5386-8088-9/19/$31.00 ©2019 IEEE

development of reinforcement learning and high computing speed of new computer, we design an autonomous agent, that perceive the feature of MEC, and the task of agent is to learn from this non-direct and delayed payoff so as to maximize the cumulative effect of subsequent actions.

Driven by solving all the aforementioned issues, in this paper, we design a reinforcement learning framework. More specifically, we propose a novel resource allocation scheme for cache-aided mobile edge computing, in which cache and computation resources are integrated for intensive computation and sensitive latency tasks. In the scheme, we formulate the considered problem as a MINLP that entails jointly optimization of task offloading decision, cache allocation, computation allocation and dynamic power distribution. We propose a MDP-based resource allocation algorithm to obtain optimal resource allocation for the AP and users. Instead of solving sophisticated joint optimization problem, the proposed algorithm is capable of learning from the past allocation experience and automatically improve allocation policy. Simulation results demonstrate that the proposed algorithm is capable of achieving a quasi-optimal performance with low computational complexity.

II. System Model

A. System Description

As is illustrated in Fig. 1, we consider a multi-user cache-aided MEC system with a single-antenna AP and N single-antenna users, denoted as $\mathcal{N} = \{1, \cdots, N\}$. The AP has affluent caching and computing capabilities at the network edge. The resources (i.e., caching resources, computing resources) are partitioned into a resource pool to provide uniform resources granularity for the system. The bandwidth between the AP and users is denoted by B. And the caching and computing capacities of the AP is denoted by C_1 and C_2, respectively. Each user has limited computation capacity and one intensive computation and sensitive latency computation task. Assuming there are M computation tasks, denoted as $\mathcal{M} = \{1, \cdots, M\}$. Each task $j \in \mathcal{M}$ is characterized by three parameters: $M_j = \{R_{i,j}, R_{w,j}, R_{r,j}\}$, where $R_{i,j}$ denotes the size of the task input (in bits), $R_{w,j}$ denotes computing capability required for this task which is quantized by the number of CPU cycles, and $R_{r,j}$ denotes the size of the computation result (in bits) [5]. We consider a scenario that the computation task may be required by multiple users in different time slots, thus caching the task computation results can improve service efficiency. The task can be computed locally or in the MEC server. The advantage of offloading computing task to the MEC server for computing is saving energy for mobile users, since the total battery capacity of mobile users is limited. However uploading the computing tasks and downloading computation results to mobile user would consume additional time and energy. The probability that user i requests for task j is denoted as $p_{i,j} \in [0, 1]$, which follows Zipf distribution as $p_{i,j} = f^j \Big/ \sum_{n=1}^{M} n^{-\delta}$, where the skew parameter δ is with typical value between 0.5 and 1.0, which determines the peakiness of the distribution [16, 17].

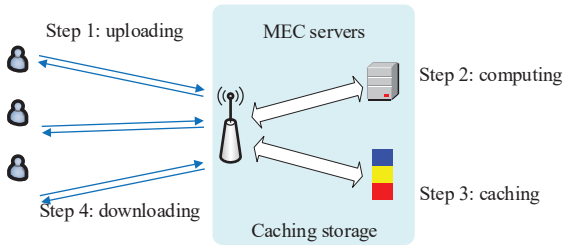

Fig. 1: An illustration of multiple-layer cache-aided mobile edge computing networks.

B. Local-Execution Computing

Assume that the CPU frequency for users is fixed. Consider a random time slot. For user i, the local computing capability (i.e. CPU cycles per second) is denoted by ω_i^l, and P_i^l denotes the energy consumption per second for local computing at this user i. If task j is computed locally, the computing during time for task j with computational requirement $R_{w,j}$ is $T_j^l = R_{w,j}/\omega_i^l$. Thus the energy consumption $E_{i,j}^{loc}$ of task j computed in user i is given by $E_{i,j}^{loc} = P_i^l R_{w,j}/\omega_i^l$.

C. MEC-Execution Computing

In case the computing tasks are offloaded to an MEC server for computing. The steps are illustrated in Fig. 1. In the first step, the computing tasks are offloaded to the AP. Let ρ_i denote the transmit power of user i to the AP in the uplink transmission. The achievable rate (in bits/s), denoted by

$$R_i = B\log_2\left(1 + \frac{\rho_i |h_i|^2 |r_i|^{-\alpha}}{\sigma^2}\right), \tag{1}$$

where σ^2 denotes the power of addictive noise power. $|r_i|^{-\alpha}$ denotes a standard distance-dependent power law pathloss attenuation between user i and the AP. In our scenario, we assume the users are randomly distributed in a square area. h_i denotes the small-scale Rayleigh fading between the AP and user i. Accordingly, the offloading time for task j with input size $R_{i,j}$ is $T_j^{offload} = R_{i,j}/R_i$. Meanwhile, the transmit energy consumption of uplink is given by $E_{i,j}^{up} = \rho_i R_{i,j}/R_i$.

Next, we illustrate the computation energy consumption at the AP. Let f_i^{mec} denotes the computing resources of the AP allocated to user i. The computing time T_j^{mec} for task j is $T_j^{mec} = R_{w,j}/f_i^{mec}$. The energy consumption per second for MEC server is denoted as P^m, thus the energy consumption $E_{i,j}^{mec}$ of task j computed in the AP is given by

$$E_j^{mec} = P^m \frac{R_{w,j}}{f_i^{mec}}. \tag{2}$$

The AP is equipped with storage resources to store reusable task computational results. Let $y_j \in \{0, 1\}$ denote whether the computation result is cached in the AP, where $y_j = 1$ if computing result of task j is cached in the AP, and $y_j = 0$ otherwise. If the computation result is cached in the AP, the AP multicasts it to all users, thus users request for same task do not need to compute it again.

The computing time for local-execution computing is denoted as T_j^l. However, the computing time T_j^{mec} for MEC-execution computing contains two parts $T_j^o = T_j^{offload} + T_j^{mec}$. The tasks computing model contains two parts, i.e., MEC-Excution Computing and Local-Excution Computing. Since the size of the task computation result is smaller than the input, and the download data rate is higher than that of the uplink, we neglect the delay and energy consumption for results downloading (like the same assumptions made in [18] [4] [19]).

The task offloading decision is denoted as $X = [x_1, x_2, \cdots, x_N]$, where $x_i \in \{0, 1\}$, $x_i = 0$ means that the task is offloaded to MEC server for computing, while $x_i = 1$ means that the task is computed locally. The computation resource allocation vector is $F^{mec} = [f_1^{mec}, f_2^{mec}, \cdots, f_N^{mec}]$, where f_i^{mec} denotes the computing resources allocated from the AP to user i. The caching vector is $Y = [y_1, y_2, \cdots, y_M]$, where $y_j = \{0, 1\}$, and $y_j \in \{0, 1\}$, where $y_j = 1$ means that the computation result of tasl j is cached in the caching storage. the transmit power vector is $P = \{\rho_1, \rho_2, \cdots, \rho_N\}$, where ρ_i denotes the transmit power of user i.

III. Problem Formulation and Proposed Approaches

A. Problem Formulation

According to the offloading decision, computation resources allocation vector, caching vector and transmit power vector, the weighted sum mobile energy consumption is given by

$$\begin{aligned} &E\left(x_i, y_i, f_i, \rho_i\right) \\ &= \sum_{i \in N} p_{i,j} \left[\left(1 - y_i\right)\left(E^{mec} + E^{up}\right) + E^{down}\right] + x_i E_{i,j}^{loc}. \end{aligned} \tag{3}$$

The offloading constraint is formulated as $\sum R_{r,n}\left(x_n = 1\right) \leq C_1$, where C_1 denotes the caching storage of the AP. The above equation shows that the task computation results cached can not exceed the total caching capacity of the AP.

The computing constraint is formulated as $\sum R_{w,n}\left(y_n = 1\right) \leq C_2$, where C_2 denotes the computing resources of the AP. The above equation shows that the task computation requirement offloaded can not exceed the total computing resources of the AP.

Also, the computing resources allocating constraint is formulated as $\sum\limits_{i \in F} f_i^{mec} \leq F$. The transmit power constraint is formulated as $\sum\limits_{i \in F} p_i \leq P_i$, where P_i denotes the total energy of user i. For task j, the computing time constrain should be satisfied.

$$\left(T_j^{offload} + T_j^{mec}\right) \leq T. \tag{4}$$

Consequently, the joint task offloading decision and resource allocation problem is formulated as below:

$$(\mathbf{P1}) \underset{X,Y,F,P}{\text{minimize}} E\left(x_i, y_i, f_j, p_j\right), \tag{5a}$$

$$\text{s.t.} \quad C_1: \; x_n \in \{0, 1\}, \forall n \in \mathcal{N}, \tag{5b}$$

$$C_2: \; y_n \in \{0, 1\}, \forall n \in \mathcal{N}, \tag{5c}$$

$$C_3: \; \sum R_{r,n}\left(x_n = 1\right) \leq C_1, \forall n \in \mathcal{N}, \tag{5d}$$

$$C_4: \; \sum R_{w,n}\left(y_n = 1\right) \leq C_2, \forall n \in \mathcal{N}, \tag{5e}$$

$$C_5: \; \sum\nolimits_{i \in N} f_i \leq F, \tag{5f}$$

$$C_6: \; \sum\nolimits_{i \in N} p_i \leq P_i, \tag{5g}$$

$$C_7: \; \left(T_j^{offload} + T_j^{mec}\right) \leq T, \forall j \in \mathcal{M} \tag{5h}$$

where x_i denotes the tasks i offloading to the AP. y_i denotes the tasks i cached in the ENs. f_j denotes the computing speed that the AP allocates to user j. p_i denotes the transmit power of user i. (5b) and (5c) guarantee the task offloading and caching is valid. (5d) guarantees the task computation results cached do not exceed the caching capacity of the AP. (5e) guarantees the task requrements offloaded do not exceed the computation resources of the AP. (5f) and (5g) guarantee the computing and power allocation is valid. (5h) guarantees the time constrain.

Lemma 1. *The formulated problem (5) is a non-convex optimization problem.*

Proof:

A close observation of problem (5), shows there are four parameters that need to be optimized, which are X, Y, F, P. The problem is non-convex due to the combinatorial selection of local computing and offloading computing mode, meanwhile the multiplicative terms in both the objective function and constrains. The details of proof is omitted for brevity. ■

The formulated optimization problem is an MINLP problem, since there are four parameters need to be optimized. To attain optimal solution of the formulated problem is non-trivial. Exhaustive search is capable of obtained optimal solution, which, however is practically infeasible due to the enormous complexity. For instance, considering a system with $N = 30$ users and $M = 50$ tasks, there will be $2^{50} = 1.13 \times 10^{15}$ different permutations, and for each permutation, the resource allocation needs to be determined.

By analyzing problem (5), we have the following lemma.

Lemma 2. *The optimal (X^*, Y^*, F^*, P^*) of problem (5) satisfies the following conditions*

$$\sum R_{r,n}\left(x_n = 1\right) = C_1. \tag{6}$$

Proof:

***Lemma 2** can be proved by using the contradictory method. Obviously, we can show that $\sum R_{r,n}\left(x_n = 1\right) = C_1$ for the optimal solution, as otherwise (5d) can be further improved by adding more task computation results into the caching storage until the equation holds. **Lemma 2** states that fully utilization of cache resources is optimal. This is intuitive since storage task computation results does not claim energy consumption.* ■

Instead of solving sophisticated joint optimization problem, we propose a low-complexity MDP-based resource allocation scheme for the formulated problem, which is capable of learning from the past allocation experience and automatically improve allocation policy.

B. MDP Model for Cache-aided MEC

In this section, we reformulate cache-aided MEC problem as a MDP model, which consists of five elements: decision epochs, states, actions, state transition probabilities, and rewards. The aim of this section is to devise a resource allocation policy function π_k that is capable of quickly generate an optimal resource allocation actions.

Definition 1. *A Markov Reward Process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. The definition of several parameters adopted in MDP are given below.*

- *State ($\mathcal{S}$): A finite set of states of the agent; System state contains four elements (x_i, y_i, f_i, p_i).*
- *Action ($\mathcal{A}$): A finite set of actions. All possibilities that the agent can do to change the states;*
- *Possibility of state transmit ($\mathcal{P}_{ss'}$): The probability that one state change into other state. For a Markov state s and successor state s', the state transition probability is defined by*

$$\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' | S_t = s, A_t = a\right]. \tag{7}$$

- *State value function ($V^{\pi}(s)$): The long-run average reward is the payoff of an infinite path where every state is assigned a real-valued reward.*

$$V^{\pi}(s) = E_{\pi}\left[\sum_{i=0}^{\infty} \gamma^i r_i \,|s = s_0\right]. \tag{8}$$

where γ is the discount factor that has the value $\gamma \in [0, 1]$. γ has critical impact on the learning process (i.e., larger γ means more importance of future return and vice versa).
We can have

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left[\sum_{i=0}^{\infty} \gamma^i r_i \,|s = s_0\right] \\ &= E_{\pi}\left[r_0 + \gamma E\left[\gamma r_1 + \gamma^2 r_2 + ...\right] |s = s_0\right] \\ &= E_{\pi}\left[r\left(s' \,|s_0, a\right) + \gamma V^{\pi}(s') \,|s = s_0\right] \end{aligned}. \tag{9}$$

- *Action value function ($V^{\pi}(s, a)$): The long-run average reward is the payoff of an infinite path where every state-action pair is assigned a real-valued reward.*

$$Q^{\pi}(s, a) = E_{\pi}\left[\sum_{i=0}^{\infty} \gamma^i r_i \,|s = s_0, a = a_0\right]. \tag{10}$$

- *A policy π is a distribution over actions given states,*

$$\pi\left(a|s\right) = P\left[A_t = a | S_t = s\right], \tag{11}$$

where a policy fully defines the behaviour of an agent. Meanwhile, MDP policies depend on the current state only.

So the optimal policy of a MDP is give as

$$\pi^*\left(s\right) = \arg\min_{\alpha} V^{\pi}(s), \forall s \in S, \tag{12}$$

which means to find optimal policy (i.e., a set of actions) t maximize the long-run average reward.

According to [20], the optimal state-value function $V^{\pi}\left(s\right)$ can be achieved by solving the Bellman's optimality equation.

Remark 1. *The optimal state-value function $V^{\pi}\left(s\right)$ satisfies the Bellman's optimality equation, which is*

$$V^{\pi}\left(s\right) = \min_{a \in A}\left\{\left(1 - \gamma\right) p\left(x, y\right) + \gamma \sum_{s' \in S} \Pr\left\{s' | s, a\right\} V\left(s'\right)\right\}. \tag{13}$$

Based on the current state $s(k)$ at time slot k, the user will decide to transmit data with power p_i. The action at time slot k is chosen from a feasible set π_k

$$\pi_k = \left\{\sum_{i \in N} f_i \leq F \wedge \sum_{i \in N} p_i \leq P\right\}. \tag{14}$$

where the symbol $\wedge$ denotes "logic AND".

Algorithm 1 MDP for resource allocation in cache-aided MEC

Initialization: Parameter γ, State s, Action a, reward of actions..
Optimal algorithm

```
Do while: Randomly initialise task offloading decision, transmit matrix and state s ;
repeat
  for each episode do
    for each step do
      Choose resource allocation action a according to policy derived from transmit matrix;
      Take the chosen action a, move state, thereafter calculate reward r of the action a and state s;
      State update: s ← s';
    end for
  end for
until state s do not change.
end for
return: task offloading matrix X, task computation result caching matrix Y, optimal policy π*_k.
```

The objective is to find optimal policy $\pi*$, which minimize the total energy consumption over a finite horizon of K time slots. Given an initial state $s(0)$, the optimal value function is calculated as

$$V* = \min_{\pi \in \prod} \sum_{k=1}^{K} \mathbb{E}\left\{E\left(k\right) |s\left(0\right), \pi\right\}, \tag{15}$$

where $\mathbb{E}$ indicates the statistical expectation with respect to the random variables. Thus the objective function is reformulated as

$$\underset{X,Y,F,P}{\text{minimize}} \sum_{k=1}^{K} E\left\{\mathbb{E}\left(k\right) |\mathbb{P}\right\}, \tag{16}$$

where $\mathbb{P}$ is the state transition probability matrix.

In order to solve this problem, Bellman's equations and backward induction are used. We develop a MDP-based resource allocation algorithm for cache-aided MEC system. In **Algorithm 1**, the discount factor, state, action are firstly initialized. Then MDP-based resource allocation algorithm is performed, which includes two cycle: the first cycle in each episode obtains the optimal policy, while the second cycle in each step attains policy in each step. The users have to decide all the actions from the beginning to the last time slot to minimize overall long-term energy consumption.

MDP solution for Cache-aided MEC is a fundamental policy update process to obtain optimal solution. However, the curse of dimensionality limits the extensiveness. In a practical scenario, the number of users and computing tasks is very large, which makes the state space and action space huge. To overcome this drawback, we adopt a model-free deep reinforcement learning scheme called deep Q networks (DQN). In DQN, the optimal policy is obtained by update Q values in a neural network. the neural network stores the whole Q-table. The update of Q-value in time slot $k+1$ in DQN is

$$Q\left(s^{k+1}, a^{k+1}\right) \leftarrow (1-\gamma)\, Q\left(s^{k}, a^{k}\right) + \alpha\left(p\left(s^{k}, a^{k}\right) + \gamma \max_{a'}\left(s^{k+1}, a\right)\right). \tag{17}$$

where α denotes the learning rate.

Remark 2. *An apparent advantage of the deep reinforcement learning algorithm is that it simplifies the problem solving. For MDP based cache-aid MEC, states and actions should be stored in the CPU units, which is a monotonous burden for limited computing units.*

Implementing our proposed deep reinforcement learning algorithm, at each epoch, the neural networks stores the state and action of MDP. And then the parameters of the networks are updated in the next epoch.

C. Complexity of the proposed algorithm

The computational complexity analysis of the proposed MDP based cache-aid MEC algorithm is evaluated as follows. There are N users and M computation tasks. The computational capacity of the MEC server and transmit power are sliced into N_f and N_P pieces, separately. The number of offloading decision for N users and caching decision for the AP are 2^N and 2^M, separately. Thus the complexity of exhaustive search for optimal solution is $O\left(2^{M+N} N^{N_f+N_P}\right)$, which is a extremely difficult task. For the proposed MDP-based cache-aid MEC algorithm, the computational complexity is $O\left(MNN_fN_P\right)$. Consequently, the proposed MDP based cache-aid MEC algorithm holds lower computational complexity than the exhaustive search.

D. Stability of the proposed algorithm

The stability of the proposed MDP based cache-aid MEC algorithm can be proved by contradiction. If there exists one more state $\hat{s}$ in the final policy π^* of the proposed MDP-based cache-aid MEC algorithm, the proposed MDP based cache-aid MEC algorithm will keep update until the state does not change, which clearly shows that the policy is not the final one.

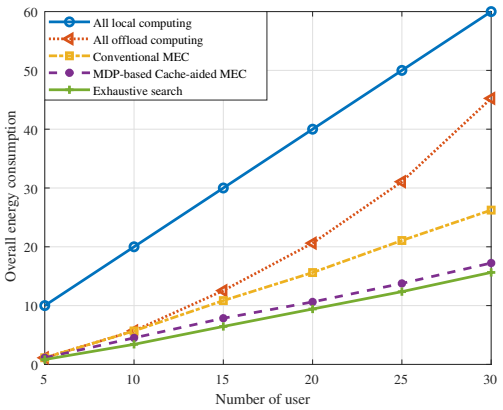

Fig. 2: Sum energy consumption vs. the number of users.

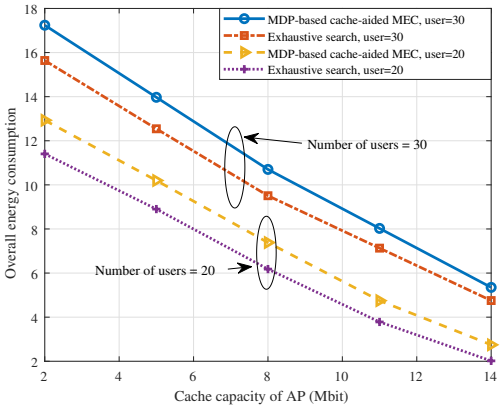

Fig. 3: Sum energy consumption vs. the caching capacity of the AP.

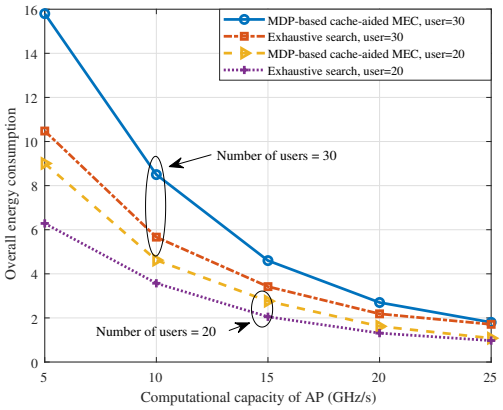

Fig. 4: Sum energy consumption vs. the computation capacity of the AP.

Therefore, the proposed MDP based cache-aid MEC algorithm is stable.

IV. Simulation Results

In this section, we present the simulation results to estimate the performance of the proposed MDP-based resource allocation algorithm for cache-aided MEC system.

In the simulation, the positions of users are randomly

distributed within a square region with length of a side 300m. The AP locates in the centre of the region. The bandwidth is 20MHz. The computation capacity of the AP is F=10GHz/sec. Hereinafter, for users, the size of task input follows the uniform distribution with $R_{i,j}(k) \in [300, 800]$ KB., meanwhile the computation resources required number of CPU cycles per bit obeys the uniform distribution with $R_{w,j}(k) \in [1000, 1500]$ cycles/bit. The CPU frequency of each user is f^l=1GHz/sec. And the computation offloading size is 500 bits. All the simulations are performed on a desktop with an Intel Core i7-7700 3.6 GHz CPU and 16 GB memory. We compare our proposed algorithm with three traditional MEC schemes: "*Full local*" means that all the tasks are computed locally in the users. "*Full offloading*" stands for that all the tasks are offloaded to the AP for computing. "*Conventional MEC*" means that there are no caching capacity in the AP.

Firstly, we evaluate the sum energy consumption against different number of users in our scenario. The sum energy consumption vs. the number of users is shown in Fig. 2. The sum energy consumption of all locally computing increase linearly. This is because the energy consumption of locally computing equals to the energy consumption of one user multiple the number of users. As can be seen from Fig. 2, the energy consumption of all offloading computing increase faster than all locally. This is because the energy consumption of all offloading computing not only consists of MEC computing consumption but also contains offloading consumption. In Fig. 2, the proposed algorithm outperforms all local computing, all offload computing and Conventional MEC. The reason is that, in cache-aid MEC, the reusable task computing results are stored in the AP, which reduce the offloading and computing energy. Also, the proposed algorithm achieves a quasi-optimal solution compared with exhaustive search.

Secondly, the overall energy consumption vs. the caching capacity of the AP is shown in Fig. 3. As can be seen from Fig. 3, the overall energy consumption decrease with the raise of caching capacity. This is because more computation results are stored in the AP, moreover, the offloading consumption and computing consumption of the same tasks required by other users are all reduced. This phenomenon is also confirmed by the insights in **Lemma 2**.

Lastly, the overall energy consumption vs. the computation capacity of the AP is shown in Fig. 4. As can be seen from Fig. 4, the overall energy consumption decrease sharply with more computational capacity. Meanwhile, compared with Fig. 3, the overall energy consumption decrease faster than increasing the cache capacity. This shows that increase the computational capacity of the AP is a more efficient way of reduce over energy consumption compared with increasing the cache capacity of the AP.

V. Conclusion

In this paper, the design of resource allocation for cache-aided MEC was considered. In order to reduce overall energy consumption of cache-aided MEC, we formulated a joint optimization problem, subjects to caching and computing resources of the AP. This problem was a MINLP problem with four parameters needed to be optimized. To attain task offloading decision of users and resource allocation for the AP, a MDP-based resource allocation algorithm scheme was proposed. Due to the large scale of states and actions space, the optimal solution was non-trivial to obtain. So we proposed a DRL algorithm with low complexity to solve the optimization problem. Simulation results demonstrated that the proposed algorithm was capable of achieving a quasi-optimal solution compared with exhaustive search. According to the simulation results, increasing computation capacity of the AP was a more efficiency method to reduce over all energy consumption compared with increasing caching capacity.

References

[1] K. Zhang, S. Leng, Y. He, and *et al.*, "Cooperative content caching in 5g networks with mobile edge computing," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 80–87, Jun. 2018.

[2] Y. Mao, C. You, J. Zhang, and *et al.*, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[3] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Int. of Things*, vol. 3, no. 6, pp. 854–864, Dec. 2016.

[4] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec 2016.

[5] C. You, K. Huang, H. Chae, and *et al.*, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, March 2017.

[6] X. Chen, H. Zhang, C. Wu, and *et al.*, "Optimized Computation Offloading Performance in Virtual Edge Computing Systems via Deep Reinforcement Learning," *ArXiv*, May 2018. [Online]. Available: http://arxiv.org/abs/1805.06146

[7] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Commun. Lett.*, vol. 6, pp. 398–401, 2017.

[8] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process. Netw*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[9] X. He, H. Xing, Y. Chen, and *et al.*, "Energy-efficient mobile-edge computation offloading for applications with shared data," *ArXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1809.00966

[10] Narendra and Fukunaga, "A branch and bound algorithm for feature subset selection," *IEEE Trans. Comput.*, vol. C-26, no. 9, pp. 917–922, Sep. 1977.

[11] D. P. Bertsekas, D. P. Bertsekas, and *et al.*, *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 2005, vol. 1, no. 3.

[12] Y. He, F. R. Yu, and *et al.*, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.

[13] M. Min, D. Xu, and *et al.*, "Learning-based computation offloading for iot devices with energy harvesting," *ArXiv*, 2017. [Online]. Available: https://arxiv.org/abs/1712.08768

[14] X. Chen, H. Zhang, and *et al.*, "Performance optimization in mobile-edge computing via deep reinforcement learning," *ArXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1804.00514

[15] L. Huang, S. Bi, and *et al.*, "Deep reinforcement learning for online offloading in wireless powered mobile-edge computing networks," *ArXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1808.01977

[16] Y. Cui, W. He, and *et al.*, "Energy-efficient resource allocation for cache-assisted mobile edge computing," in *in IEEE Proc. of Loc. Com. Netw. (LCN)*, Oct 2017, pp. 640–648.

[17] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," in *Proc. Int.Conf.Comput.Commun. (INFOCOM)*, vol. 1, Mar. 1999, pp. 126–134 vol.1.

[18] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Commun. Technol.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.

[19] C. You, K. Huang, and H. Chae, "Energy efficient mobile cloud computing powered by wireless energy transfer," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1757–1771, May 2016.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2016.