Dynamic Task Caching and Computation Offloading for Mobile Edge Computing

Zhixiong Chen*, and Zhaokun Zhou†
*School of Microelectronics and Communication Engineering,
Chongqing University, Chongqing 400044, P. R. China
†Chongqing Automotive Collaborative Innovation Center,
Chongqing University, Chongqing 400044, P. R. China
Emails: {czx,zhaokunzhou}@cqu.edu.cn

GLOBECOM 2020 - 2020 IEEE Global Communications Conference | 978-1-7281-8298-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/GLOBECOM42002.2020.9322644

***Abstract*—Mobile edge computing (MEC) provides information technology and cloud-computing capabilities within the network edge close to mobile users, thereby addressing computing demand for users. However, the energy consumption of data uploading from users to the MEC server makes it hard to meet users' demand in some specific applications, i.e., interactive gaming and augmented reality. Motivated by this, we integrate a task caching mechanism into computation offloading technique. Specifically, it allows the MEC server to proactively cache some tasks and users to offload their tasks to the MEC server. Since the limited storage capacity and task demands for users are changing dynamically, which tasks are cached has to be judiciously decided to maximize the MEC system performance. The objective of this paper is to minimize the system cost, which is defined as the average total user energy consumption of all time slots. By formulating the problem as an integer-programming, we propose to find the optimal solution with two steps. Through which we have obtained the optimal online computation offloading and task cache update strategy. Simulation results show that in comparison with the other two baselines, the proposed scheme can effectively reduce the system cost.**

***Index Terms*—Mobile edge computing, computation offloading, caching.**

I. Introduction

The increasing computation-intensive and data-hungry tasks, i.e., interactive gaming and virtual/augmented reality (VR/AR) become more and more popular. This brings a huge challenge for resource-constrained mobile user devices. To alleviate computing requirement of users, mobile edge computing was firstly proposed by the European Telecommunications Standard Institute (ETSI) [1] in 2014 and was defined as a new platform that provides computing services within the radio access network in close proximity to mobile subscribers. The authors in [2] discussed multi-user multi-task offloading scheduling schemes in a renewable mobile edge cloud system. A holistic strategy for a joint task offloading and resource allocation in a multi-cell Mobile edge computing (MEC) network was proposed to maximize the reductions in task completion time and energy consumption [3]. In [4], the authors developed resource allocation schemes for a multiuser MEC system based on time-division multiple access (TDMA) and orthogonal frequency-division multiple access (OFDMA).

With the emergence of more data-hungry and sophisticated application, data caching was proposed to further improve the computing performance for mobile users. Especially, the authors in [5] investigated the computation capability and content caching problem, which aim to improve quality of experience (QoE) of mobile users with latency-sensitive computation tasks. In [6], the authors proposed a joint service caching and task offloading scheme for MEC-enabled dense cellular networks to reduce computation latency. The authors in [7] studied joint service placement and request routing in MEC-enabled multi-cell networks. In [8], the authors explored the problem of joint computational offloading and content caching in the wireless heterogeneous MEC system. The authors in [9] focused on the heterogeneity of edge node characteristics and user locations, they investigated the problem of placing multiple services in the heterogeneous MEC system to maximize the total reward.

However, all the above literature is statically caching data or contents at the network edge, they prefer to the caching policies under which content placement remains unchanged over a relatively long time. In fact, users' demand for computation tasks is dynamically changing over time. Motivated by this, we consider to employ dynamic task caching technique at an MEC-enabled cellular network. In particular, the task caching and computation offloading policies are dynamically changed by making full use of instantaneous user computation demand. This makes the tasks cached by the MEC server more likely to be utilized by users and the performance of the MEC system can be further improved. In this paper, we formulated the problem as an integer programming problem that involves storage capacity, user demand and task caching constraints. Through some detailed analysis, we first propose an online optimal computation offloading scheme. Then, we develop a dynamic programming-based algorithm to address the online optimal task cache update strategy. Finally, the optimal joint online computation offloading and task cache update strategy is derived to obtain the optimal performance of the proposed scheme. By comparing the proposed scheme with other two baselines, it is found that the proposed dynamic task caching and computation offloading scheme can effectively reduce the energy consumption for users.

978-1-7281-8298-8/20/$31.00 ©2020 IEEE

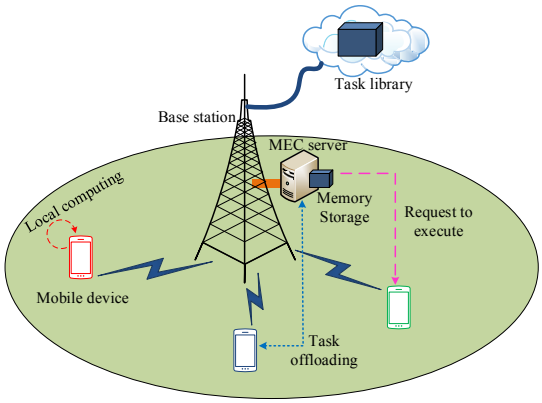


Fig. 1. MEC-enabled Cellular Network System.

The rest of this paper is organized as follows. In Section II, we introduce the system model and formulate the system cost minimization problem. In Section III, we propose an efficient scheme to solve the original problem. Section IV verifies the effectiveness of the proposed scheme by simulations. The conclusion is drawn in Section V.

II. System Model

A. Network Model

We consider an MEC-enabled cellular network, as shown in Fig. 1. There is one base station (BS) in the network serving for K users, whose index set is denoted by $\mathcal{K} = \{1, 2, \cdots, K\}$. The BS is equipped with an MEC server to provide computing offloading and storage services to the users such as phones, wearable devices and tablets. The MEC server can access to a task library from cloud that contains F tasks, denoted by $\mathcal{F} = \{1, 2, \cdots, F\}$. Specifically, each task $f \in \mathcal{F}$ is characterized by a tuple of two parameters. The first one is the total number of CPU cycles required to accomplish task f, denoted by S_f. The second one identifies the size of input data necessary to transfer the task execution for task f, denoted by D_f, i.e. program codes and input parameters. Thus, we can describe task f as $Z_f \triangleq \langle S_f, D_f \rangle$. Let $b_f^{(t)} \in \{0, 1\}$ denote the cache state of task f at the MEC server storage space at time slot t, where $b_f^{(t)} = 1$ indicates that task f is stored at the MEC server and $b_f^{(t)} = 0$ otherwise. Thus, the cache state of the MEC server at time slot t is represented by $\boldsymbol{b}_t = \{b_f^{(t)} : f \in \mathcal{F}\} \in \{0, 1\}^F$. Denoted by C the storage space size of the MEC server. We have

$$\sum_{f \in \mathcal{F}} b_f^{(t)} \cdot D_f \leq C. \tag{1}$$

The system operates over an infinite time horizon and time is slotted by equal length τ, indexed by $t = 0, 1, 2, \cdots$. At the beginning of each time slot, each user submits at most one task execution request, which is assumed to be delay intolerant and must be accomplished before the end of the slot, either by its own computing or by the MEC server execution. At time slot t, let $u_k^{(t)} \in \overline{\mathcal{F}} = \{0\} \cup \mathcal{F}$ denote the task request state of user k, where $u_k^{(t)} = 0$ indicates that user k requests nothing. $u_k^{(t)} = f \in \mathcal{F}$ indicates that user k requests to execute task f. Note that $\overline{\mathcal{F}}$ is the demand state space of each user which is of cardinality $F + 1$. Sequentially, we use $\boldsymbol{u}_t = \{u_k^{(t)} : k \in \mathcal{K}\} \in \overline{\mathcal{F}}^K$ to denote the user demand state of the system. Besides, we assume that $u_k^{(t)}$ evolves according to a first-order $(F + 1)$-state Markov chain, which captures both task popularity and inter-task correlation of order one of user k's demand probability. Let $\Pr[u_k^{(t+1)} = j \,|\, u_k^{(t)} = i]$ denote the transition probability of going to request task $j \in \overline{\mathcal{F}}$ at time slot $t + 1$ given that the request state at time slot t is $i \in \overline{\mathcal{F}}$. In this paper, we assume that $u_k^{(t)}$ is time-homogenous. We denote the task request transition probability matrix of $u_k^{(t)}$ is $Q_k = \left[q_{i \to j}^{(k)}\right]_{i,j \in \overline{\mathcal{F}}}$. Similar as [10], $q_{i \to j}^{(k)} = \Pr[u_k^{(t+1)} = j \,|\, u_k^{(t)} = i]$ is given by

$$q_{i \to j}^{(k)} = \begin{cases} R_k, & i \in \overline{\mathcal{F}}, j = 0, \\ (1 - R_k) \frac{\frac{1}{j^{\gamma_k}}}{\sum_{n=1}^{F} \frac{1}{n^{\gamma_k}}}, & i = 0, j \in \mathcal{F}, \\ (1 - R_k) \frac{1}{N_k}, & i \in \mathcal{F}, j = (i + n) \text{mod} (F + 1), \\ & n \in \{1, 2, \cdots, N_k\}, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

$q_{i \to j}^{(k)}$ is parameterized by $\langle R_k, \gamma_k, N_k \rangle$. Specifically, R_k is the transition probability of requesting nothing given any task request at the current time slot. The transition probability of any task $f \in \mathcal{F}$ under no current file request is modeled as a Zipf distribution which parameterized by γ_k. Besides, for any task $i \in \mathcal{F}$, we assign a set of N_k neighboring tasks, i.e., $\mathcal{N}_k = \{f \in \mathcal{F}, f = (i + n) \mod (F + 1), n = 1, 2, \cdots N_k\}$. Then, the transition probability of requesting any task $f \in \mathcal{N}_k$ under the current task request $i \in \mathcal{F}$ is modeled as the uniform distribution. The transition probability of requesting any task $f \notin \mathcal{N}_k$ under the current task request $i \in \mathcal{F}$ is zero.

B. Local Computing

When user k chooses to execute its requested task by local CPU, we denote the computing capability (i.e. CPU cycles per second) of user k ($k \in \mathcal{K}$) as f_k^{L}. Employing dynamic voltage and frequency scaling (DVFS) techniques [11], user k can control the energy consumption for local computing by adjusting the CPU frequency. Since user k requires to finish local task computing within the current time slot with length τ. Thus, we have

$$f_k^{\text{L}} = \frac{S_f}{\tau}. \tag{3}$$

Since the energy consumption is proportional to the square of the frequency of mobile device [12], we can express the corresponding energy consumption when task f is executed locally as follows.

$$E_{k,f}^{\text{L}} = \zeta (f_k^{\text{L}})^2 S_f = \zeta \frac{S_f^3}{\tau^2}, \tag{4}$$

where ζ is the energy coefficient, which depends on chip architecture.

C. Computation Offloading

When user k offloads computation task f to the MEC server to execute, it will upload the necessary input data to the MEC server for transferring the task execution. In this paper, we assume that the network is based on the orthogonal frequency division multiple-access (OFDMA) technique in which K users are separated in the frequency domain with equal size bandwidth B. Note that, using such transmission for the uplink offloading implies that the users do not interfere with one another. Let P_k denote the transmission power of user k. Denote by H_k the channel gain between the user k and the BS. In this paper, we assume that the user mobility is not high during the task offloading duration, thus H_k is a constant due to low user mobility and quasi-static fading. Consequently, the achievable uplink rate for user k is

$$r_k = B\log(1+\frac{P_k H_k}{\sigma^2}), \tag{5}$$

where σ^2 is the variance of complex white Gaussian channel noise.

Let $\left(f_{\mathrm{C}} \gg f_k^{\mathrm{L}}, \forall k \in \mathcal{K}\right)$ represents the computational capability of the MEC server. When user k offloads task f to the MEC server for executing, the total delay can be expressed as follows.

$$T_{k,f}^{\mathrm{O}} = \frac{D_f}{r_k} + \frac{S_f}{f_{\mathrm{C}}}, \tag{6}$$

where the first part is the transmission delay when user k offloads task f, the last part is the computing delay of task f processed on the MEC server. Similar to many studies such as [3] and [4], we ignore the transmission delay for the MEC server to send the task results back to the users. This is because the result data size is generally small, and downlink rate from BS to mobile devices is higher than the uplink rate from mobile devices to BS. Besides, the energy consumed by user k when it offloads its task to the MEC server to execute can be expressed as

$$E_{k,f}^{\mathrm{O}} = P_k \times \frac{D_f}{r_k}. \tag{7}$$

Denote with $\alpha_{k,f}^{(t)} \in \{0,1\}$ the task offloading decision of user k accomplish task f, where $\alpha_{k,f}^{(t)} = 1$ indicates that user k decides to offload task f to the MEC server and $\alpha_{k,f}^{(t)} = 0$ otherwise. We use $\boldsymbol{\alpha}_t = \left\{\alpha_{k,f}^{(t)} : k \in \mathcal{K}, f \in \mathcal{F}\right\}$ represents task offloading decisions for all users at time slot t. It is reasonable that users do not offload cached tasks. Thus, we have

$$\alpha_{k,f}^{(t)} \leq 1 - b_f^{(t)}, \forall k \in \mathcal{K}, \forall f \in \mathcal{F}. \tag{8}$$

It is straightforward to see that user k does not offload the non-requested task of itself. Thus, it has

$$\alpha_{k,f}^{(t)} \leq \mathbb{1}(u_k^{(t)} = f), \forall k \in \mathcal{K}, \forall f \in \mathcal{F}, \tag{9}$$

where $\mathbb{1}(\cdot)$ is an indicator function, which is one if and only if the condition in parentheses is true, otherwise is zero.

Besides, the user requested task must be accomplished in the current time slot. Thus, the task offloading action satisfies the following constraint.

$$\alpha_{k,f}^{(t)} T_{k,f}^{\mathrm{O}} + \left(1 - \alpha_{k,f}^{(t)}\right) b_f^{(t)} T_{k,f}^{\mathrm{C}} \leq \tau, \forall k \in \mathcal{K}, \forall f \in \mathcal{F}. \tag{10}$$

D. Request to Execute Cached Tasks

In this paper, we introduce an efficient way to process computational tasks, which is to pre-cache a part of tasks (such as codes) at the MEC server. When a user needs to execute task f, it can first check whether the MEC server stores task f. Once f is confirmed in the MEC server storage space, the user can request the MEC server to execute task f directly without uploading procedure, then the MEC server returns the computational results of task f to it. According to the above MEC computing method, the delay for user k request to execute task f can be expressed as follows.

$$T_{k,f}^{\mathrm{C}} = \frac{S_f}{f_{\mathrm{C}}}. \tag{11}$$

In this case, we consider the energy consumption of user k is zero. We neglect the receive energy consumption of devices due to the receive energy consumption is generally smaller than the energy consumption of other parts.

After the requested task being executed by the MEC server, the server updates its cache state. Let $\beta_f^{(t)} \in \{-1, 0, 1\}$ denote the caching update decision for task f on the MEC server at the end of time slot t, where $\beta_f^{(t)} = -1$ means that task f is removed from the storage space of the MEC server, $\beta_f^{(t)} = 0$ implies that task f at the MEC server does not changed, $\beta_f^{(t)} = 1$ indicates that task f is cached at the MEC server at time slot t. We use $\boldsymbol{\beta}_t = \left\{\beta_f^{(t)} : f \in \mathcal{F}\right\}$ denote the caching action of the MEC server at time slot t. It is straightforward to realize that the MEC server cannot remove the non-cached tasks. Thus, the caching update decisions satisfy the following constraint.

$$\beta_f^{(t)} \geq -b_f^{(t)}, \forall f \in \mathcal{F}. \tag{12}$$

The caching state evolves according to

$$b_f^{(t+1)} = b_f^{(t)} + \beta_f^{(t)}, \forall f \in \mathcal{F}. \tag{13}$$

Since $b_f^{(t+1)} \in \{0,1\}$ and satisfies the storage capacity of the MEC server, we have the following constraints.

$$b_f^{(t)} + \beta_f^{(t)} \in \{0,1\}, \forall f \in \mathcal{F}, \tag{14}$$

$$\sum\nolimits_{f \in \mathcal{F}} (b_f^{(t)} + \beta_f^{(t)}) \cdot D_f \leq C. \tag{15}$$

E. Problem Formulation

In this paper, for the sake of reducing the task execution energy consumption, we consider to optimize the joint computation offloading and task update caching action, i.e. $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$. Based on the above analysis, we can obtain the close-form

expression of energy consumption when user k executes task f as follows.

$$E_{k,f}^{(t)} = \alpha_{k,f}^{(t)} E_{k,f}^{\mathrm{O}} + \left(1 - \alpha_{k,f}^{(t)}\right)\left(1 - b_f^{(t)}\right) E_{k,f}^{\mathrm{L}}. \quad (16)$$

So far, we can formulate the optimal system cost problem as follows.

$$\mathcal{P}: \quad \min_{\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t} \quad \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{k\in\mathcal{K}} \sum_{f\in\mathcal{F}} \mathbb{1}\left(u_k^{(t)} = f\right) E_{k,f}^{(t)} \quad (17)$$

$$\text{s. t.} \quad (8), (9), (10), (12), (15), \quad (17a)$$

$$\alpha_{k,f}^{(t)} \in \{0,1\}, \forall k \in \mathcal{K}, \forall f \in \mathcal{F}, \quad (17b)$$

$$\beta_f^{(t)} \in \{-1, 0, 1\}, \forall f \in \mathcal{F}. \quad (17c)$$

III. Online computation offloading and task update strategy

In this section, by observing the structure of problem $\mathcal{P}$, we obtain the optimal computation offloading strategy and task update decision by decomposing $\mathcal{P}$ into two subproblems and solve them one by one.

A. Optimal Computation Offloading

The major challenge of directly solving problem $\mathcal{P}$ is that the long-term interactive computation offloading and task caching across different time slots. To address this challenge, we first analyze the relationship of offloading and caching decisions in different time slots. Then, we leverage the dependency between decision variables and system cost of different time slots to solve the optimal computation offloading strategy and task cache update decision.

Problem $\mathcal{P}$ can be described as an infinite horizon average cost MDP. The system state consists of the cache state and task request state of users. At time slot t, denote with $\mathcal{X}_t = (\boldsymbol{b}_t, \boldsymbol{u}_t) \in \{0,1\}^F \times \overline{\mathcal{F}}^K$ the system state, where $\{0,1\}^F \times \overline{\mathcal{F}}^K$ represents the system state space. The system action consists of both computation offloading decision and task cache update decision, denoted as $\mathcal{A}_t = (\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t) \in \{0,1\}^K \times \{-1,0,1\}^F$, where $\{0,1\}^K \times \{-1,0,1\}^F$ represents the system action space. The system cost at time slot t can be characterized as the sum of energy consumption of all users, that is $\sum_{k\in\mathcal{K}} \sum_{f\in\mathcal{F}} \mathbb{1}(u_k^{(t)} = f) \cdot E_{k,f}^{(t)}$. Besides, the transition probability of the system state is given by

$$\begin{aligned} \Pr[\mathcal{X}_{t+1} \,|\mathcal{X}_t\,, \mathcal{A}_t] &= \Pr[(\boldsymbol{b}_{t+1}, \boldsymbol{u}_{t+1}) \,|(\boldsymbol{b}_t, \boldsymbol{u}_t)\,, (\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t)] \\ &= \Pr[\boldsymbol{b}_{t+1} \,|\boldsymbol{b}_t\,, \boldsymbol{\beta}_t] \times \Pr[\boldsymbol{u}_{t+1} \,|\boldsymbol{u}_t\,] \\ &= \begin{cases} \prod\limits_{k\in\mathcal{K}} q_{u_k^{(t)} \to u_k^{(t+1)}}^{(k)}, & \boldsymbol{b}_{t+1} = \boldsymbol{b}_t + \boldsymbol{\beta}_t, \\ 0, & \boldsymbol{b}_{t+1} \neq \boldsymbol{b}_t + \boldsymbol{\beta}_t. \end{cases} \end{aligned} \quad (18)$$

Based on the above transition probability equation, it is straightforward to see that the task offloading action does not affect the system state. In other words, the computation offloading decision just affects the system cost of the current time slot. It does not affect the system state or system cost of other time slots. However, the computation offloading decision is restricted by the current cache state and task request state. Motivated by this, for any given cache state and task request state at time slot t, we can establish the optimal task offloading decision online. For clarity, we formulate the computation offloading problem as follows.

$$\mathcal{P}_1: \quad \min_{\boldsymbol{\alpha}_t} \quad f(\boldsymbol{\alpha}_t) \quad (19)$$

$$\text{s. t.} \quad (8), (9), (10), \quad (19a)$$

$$\alpha_{k,f}^{(t)} \in \{0,1\}, \forall k \in \mathcal{K}, \forall f \in \mathcal{F}, \quad (19a)$$

where

$$f(\boldsymbol{\alpha}_t) = \sum_{k\in\mathcal{K}} \sum_{f\in\mathcal{F}} \Big\{ \alpha_{k,f}^{(t)} E_{k,f}^{\mathrm{O}} + (1 - \alpha_{k,f}^{(t)})(1 - b_f^{(t)}) E_{k,f}^{\mathrm{L}} \Big\} \mathbb{1}\left(u_k^{(t)} = f\right). \quad (20)$$

For problem $\mathcal{P}_1$, the task offloading actions of users are independent with each other. Each user can make computation offloading decision based on its demand and the cache state of the MEC server. Thus, we can optimize the task offloading decision of each user lonely. Since the offloading decision is a binary variable, we can directly compare the energy consumption under offloading and no-offloading and select the minimal energy consumption decision. Thus, we can easily obtain the optimal computation offloading strategy.

B. Optimal Task Cache Update

Based on the above discussions, we can obtain the optimal computation offloading strategy for any given system state, i.e., task cache state and user request state. Thus, we can express the energy consumption of users under any task cache state and user request state. Furthermore, we can evaluate the energy consumption for any task cache update decision. Based on some detailed analysis, we propose an online optimal task cache update algorithm to select the optimal cache update action in each time slot.

At time slot t, after the tasks of users being executed, the system state can be updated. Then, the system transfers to the next time slot and select the optimal computation offloading strategy to accomplish users' requested tasks. Thus, the cache update decision in the current time slot just affects the system cost in the next time slot. Assume that at time slot $t+1$, the system will select the optimal computation offloading strategy based on the cache state and task request state. We can formulate the optimal cache update problem at time slot t as follows.

$$\mathcal{P}_2: \quad \min_{\boldsymbol{\beta}_t} \quad h(\boldsymbol{\beta}_t) \quad (21)$$

$$\text{s. t.} \quad (12), (15), \quad (21a)$$

$$\beta_f^{(t)} \in \{-1, 0, 1\}, \forall f \in \mathcal{F}, \quad (21b)$$

where

$$h(\boldsymbol{\beta}_t) = \sum_{k\in\mathcal{K}} \sum_{f\in\mathcal{F}} \Big\{ \alpha_{k,f}^{(t+1)} E_{k,f}^{\mathrm{O}} + \left(1 - \alpha_{k,f}^{(t+1)}\right)\left(1 - b_f^{(t+1)}\right) E_{k,f}^{\mathrm{L}} \Big\} \mathbb{1}\left(u_k^{(t+1)} = f\right). \quad (22)$$

Since the task requested transition probability is known, we can obtain the requested probability of time slot $t+1$ under the given requested state at time slot t. Motivated by this, we can find the optimal task update strategy at time slot t by using dynamic programming. For the sake of solving problem $\mathcal{P}_2$, we first find the optimal system state at time slot $t+1$, then we obtain the optimal task update decision based on Eq. (13). Thus, we can reformulate problem $\mathcal{P}_2$ as the following equivalent problem.

$$\widetilde{\mathcal{P}_2}: \quad \min_{\boldsymbol{b_t}} \quad \sum_{f\in\mathcal{F}}\sum_{k\in\mathcal{K}} q_{u_k^{(t)}\to f}\{\alpha_{k,f}^{(t+1)}E_{k,f}^{\mathrm{O}} + (1-\alpha_{k,f}^{(t+1)})(1-b_f^{(t+1)})E_{k,f}^{\mathrm{L}}\} \tag{23}$$

$$\text{s. t.} \quad (12),(15), \tag{23a}$$

$$b_f^{(t)} \in \{0,1\}, \forall f \in \mathcal{F}. \tag{23b}$$

For problem $\widetilde{\mathcal{P}_2}$, the optimal task cache state can be derived when the cache size is given. Below we introduce a recursive function to derive the optimal solution. For ease of presentation, we first define a $F \times C$ matrix W, in which $W(f,q)$ represents the optimal solution under the first f tasks using a cache size of q. The value of $W(f,q)$ is given by the following recursive function.

$$W(f,q) = \min_{b_f^{(t+1)}} \Big\{ W(f-1, q-b_f^{(t+1)}D_f) + \sum_{k\in\mathcal{K}} q_{u_k^{(t)}\to f}^{(k)} \Big[\alpha_{k,f}^{(t+1)}E_{k,f}^{\mathrm{O}} + (1-\alpha_{k,f}^{(t+1)})(1-b_f^{(t+1)})E_{k,f}^{\mathrm{L}}\Big]\Big\}. \tag{24}$$

Through the above recursive function, the optimal system state at time slot $t+1$ is given by the argument of $W(F,C)$. Hence, we can solve the optimal task cache update strategy at time slot t. For clarity, we conclude the detailed steps of the task cache update algorithm as Algorithm 1. According to the recursive function (24), the time complexity of the proposed task cache update algorithm is $\mathcal{O}(2FC+F)$.

Algorithm 1 Algorithm for Task Cache Update.

Input: $C, F, K, [D_f, S_f : \forall f \in \mathcal{F}], u_t, r, P_k, \zeta, \tau, f_{\mathrm{C}}, R, N, \gamma$
Output: Task update decision $\boldsymbol{\beta}_t$

$b_{t+1} = [0]_F, W = [0]_{F\times C}, W_r = [0]_{F\times C}$;
for each $f \in [1, F]$ **do**
  if $f < F$ **then**
    for each $q \in [1, C]$ **do**
      if $f == 1$ **then**
        $W_r(f,q) = \mathbb{1}(D_f < q) \arg\min_{b_f^{(t+1)}} \sum_{k\in\mathcal{K}} q_{u_k^{(t)}\to f} \times \{\alpha_{k,f}^{(t+1)}E_{k,f}^{\mathrm{O}} + (1-\alpha_{k,f}^{(t+1)})(1-b_f^{(t+1)})E_{k,f}^{\mathrm{L}}\}$
        $W(f,q) = \sum_{k\in\mathcal{K}} q_{u_k^{(t)}\to f}\{\alpha_{k,f}^{(t+1)}E_{k,f}^{\mathrm{O}} + (1-\alpha_{k,f}^{(t+1)})(1-W_r(f,q))E_{k,f}^{\mathrm{L}}\}$
      else
        $W_r(f,q) = 1(D_f < q) \arg\min_{b_f^{(t+1)}} \Big\{ W(f-1, q-b_f^{(t+1)}D_f) + \sum_{k\in\mathcal{K}} q_{u_k^{(t)}\to f}\Big(\alpha_{k,f}^{(t+1)}E_{k,f}^{\mathrm{O}} + (1-\alpha_{k,f}^{(t+1)})(1-b_f^{(t+1)})E_{k,f}^{\mathrm{L}}\Big)\Big\}$
        $W(f,q) = W(f-1, q-W_r(f,q)D_f) + \sum_{k\in\mathcal{K}} q_{u_k^{(t)}\to f}\Big\{\alpha_{k,f}^{(t+1)}E_{k,f}^{\mathrm{O}} + (1-\alpha_{k,f}^{(t+1)})(1-W_r(f,q))E_{k,f}^{\mathrm{L}}\Big\}$
  else
    $W_r(F,C) = 1(D_f < q) \arg\min_{b_f^{(t+1)}} \Big\{ W(F-1, C-b_f^{(t+1)}D_f) + \sum_{k\in\mathcal{K}} q_{u_k^{(t)}\to f}\Big(\alpha_{k,f}^{(t+1)}E_{k,f}^{\mathrm{O}} + (1-\alpha_{k,f}^{(t+1)})(1-b_f^{(t+1)})E_{k,f}^{\mathrm{L}}\Big)\Big\}$
    $W(F,C) = W(F-1, C-W_r(F,C)D_f) + \sum_{k\in\mathcal{K}} q_{u_k^{(t)}\to f}\Big\{\alpha_{k,f}^{(t+1)}E_{k,f}^{\mathrm{O}} + (1-\alpha_{k,f}^{(t+1)})(1-W_r(F,C))E_{k,f}^{\mathrm{L}}\Big\}$
$b_{t+1}(F) = W_r(F,C)$;
for each $f = F-1 : -1 : 1$ **do**
  $b_{t+1}(f) = W_r(f, C - \sum_{f+1\le j\le F} b_t(j) * D_j)$;
Compute $\boldsymbol{\beta}_t$ based on Eq. (13);

IV. Simulation Results

In this section, we evaluate the proposed computation offloading and task cache update algorithm by comparing its performances with other two benchmark schemes. The first one is an MEC offloading scheme where the task caching is not considered. For evaluating the performance of the proposed scheme, we improve the least recently used (LRU) policy and the least frequently used (LFU) policy [13] to construct a new scheme called LRFU as the second benchmark, which cache the least recently and frequently used task.

A. Simulation Setup

In the simulations, we consider a single cell that K users are randomly distributed over a $200m \times 200m$ region and the base station is located at the center of the region. The uplink transmission bandwidth is set to $B = 10$MHz and the transmission power of all users are set to $P_k = 0.5W$. According to the realistic measurements in [12], we set the energy coefficient ζ as 5×10^{-27}. The white Gaussian noise variance is assumed to be $\sigma^2 = 2 \times 10^{-13}$. In addition, the channel gain is modeled as $H_k = 127 + 30 \times \log d_k$, where d_k is the distance between user k and the BS. The input data size of each task, i.e., D_f, is uniform randomly selected in $[1, 4]$ Gigabytes. The required CPU cycles for computing task k, i.e., S_f, is randomly selected in $[5, 10]$ Gigacycles. In terms of computing resources, we set the CPU capability of the MEC server to be 5GHz. Besides, the task request parameters for all users are set to be homogeneous, that is $R_k = 0.2$, $N_k = 10$ and $\gamma = 0.8$.

B. Simulation Analysis

Fig. 2 shows the system cost of the proposed and two benchmark schemes under a different number of users. It is seen that along with the increases in user number, the system cost of the three schemes keeps increasing. This is because the storage and computing resource of the MEC server is limited. It is also observed in Fig. 2 that the system cost

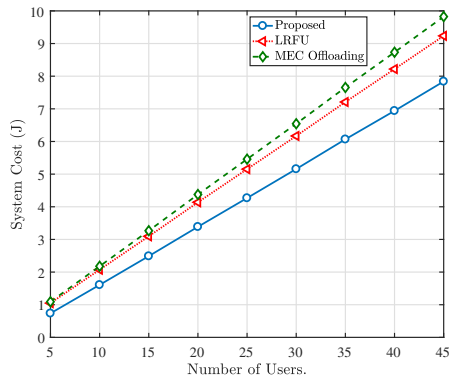
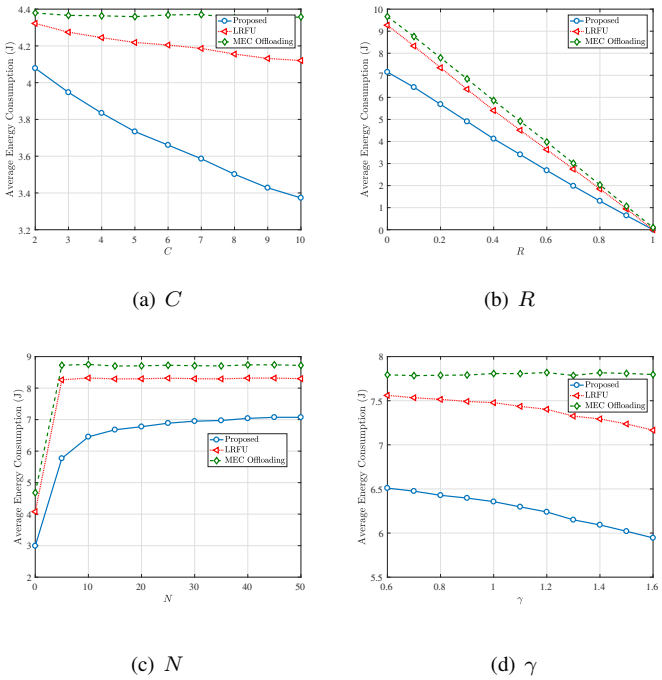


Fig. 2. Comparison of system cost against different user number. $F = 100$.

of our proposed scheme is significantly suppressed compared with the other three benchmark schemes.

The relationship between the average energy consumption and the cache size of the MEC server is illustrated in Fig. 3(a). It can be seen that the system cost of the proposed and LRFU scheme is decreasing along with the increase of capacity value. This is because along with the increase of C, users get more low-cost resources to handle their tasks. When C is large enough, the MEC server can store all contents, the energy consumption curve of the proposed scheme coincides with the LRFU. Similar to other simulation results, the performance of the proposed scheme is better than the other two benchmarks.



(a) C (b) R

(c) N (d) γ

Fig. 3. Comparison of the proposed and the two baselines.$K = 10$, $F = 100$, unless otherwise stated.

Fig. 3(b)-3(d) illustrate the user average energy consumption versus the parameters of the transition matrix of each user's task request process, i.e., R, N and γ. Specifically, Fig. 3(b) show how the user average energy consumption vary with R. It can be clearly seen that along with the increase of R, the user average energy consumption keeps decreasing. This is because the task demand becomes light. From Fig. 3(c), the user average energy consumption increases with N, since the user demand processes become less predictable as N becomes larger. From Fig. 3(d), we see that the user average energy consumption of the proposed and LRFU scheme monotonically decreases with γ. This is because along with the increase of γ, the skewness of the task requested probability increases. Most of the user requests concentrate in a few tasks and the majority of the tasks in the library have a very low probability of being requested.

V. Conclusion

In this paper, we have investigated the optimal computation offloading and task caching update problem in MEC-enabled multi-user wireless cellular network. The optimization problem is formulated as an integer-programming problem. Based on our detailed analysis, we find the optimal solution through two steps. Through which we have obtained the optimal computation offloading strategy and a dynamic programming-based algorithm yields the optimal task update decision. It is shown by simulations that the proposed scheme can significantly decrease the system cost.

References

[1] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, *et al.*, "Mobile edge computinga key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[2] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Trans. Services Compu.*, 2018.

[3] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, 2018.

[4] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, 2016.

[5] J. Zhang, X. Hu, Z. Ning, E. C.-H. Ngai, *et al.*, "Joint resource allocation for latency-sensitive services over mobile edge computing networks with caching," *IEEE Internet Things J.*, 2018.

[6] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE INFOCOM*, 2018, pp. 207–215.

[7] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, *et al.*, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. IEEE INFOCOM*, 2019, pp. 10–18.

[8] A. Asheralieva, "Optimal computational offloading and content caching in wireless heterogeneous mobile edge computing systems with hopfield neural networks," *IEEE Trans. Emerging Topics Comput. Intell.*, 2019.

[9] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *Proc. IEEE INFOCOM*, 2019, pp. 514–522.

[10] K. Psounis, A. Zhu, B. Prabhakar, and R. Motwani, "Modeling correlations in web traces and implications for designing replacement policies," *Comput. Net.*, vol. 45, no. 4, pp. 379–398, 2004.

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

[12] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing." *HotCloud*, vol. 10, no. 4-4, p. 19, 2010.

[13] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for lru cache performance," in *Proc. IEEE ITC*, 2012, pp. 1–8.