Dependent Task offloading and Service Caching with State Management for Mobile Edge Computing

Zhi Ma†, Sheng Zhang*†, Ning Chen†, Zhuzhong Qian†, Qing Gu†, Yu Liang‡ and Sanglu Lu†
†State Key Lab. for Novel Software Technology, Nanjing University, CN
‡ School of Computer and Electronic Information/School of Artificial Intelligence, Nanjing Normal University, CN
Email: sheng@nju.edu.cn

***Abstract*—The widespread use of 5G and artificial intelligence applications has led to strong momentum in Mobile Edge Computing (MEC). With MEC, we can offload compute-intensive tasks to edge servers that are closer to the user, thereby reducing the long latency incurred by data transmission via WAN. Although many works have investigated task offloading decisions under service caching, the state of services is an equal, if not more important, research area of MEC, yet receive much less attention. In general, the arrival of tasks exhibit a distribution over time. Besides the necessary energy consumption in processing tasks offloaded to edge servers, a large amount of energy is required for maintaining services cached on servers. When more and more services become idle, they will incur a non-negligible additional energy. In this paper, we focus on an interesting but currently less studied problem in MEC, namely online service caching and state management in MEC. We propose DCSO, a bounded online algorithm that considers dynamic service caching and state management of services to minimize long-term cost in MEC systems. Meanwhile, our algorithm achieves a 2 competitive ratio in state management. Trace-driven simulations show that our algorithm reduces the overall cost efficiently while keeping low computation latency.**

I. Introduction

Nowadays, the rapid development of 5G and Artificial Intelligence (AI) has led to the emergence of a large number of computation-intensive applications (e.g., augmented/virtual reality [1, 2], and video processing [3]). These compute-intensive applications require large amounts of computing and storage resources that most smart devices cannot provide. Meanwhile, the traditional way of offloading computation tasks to the cloud brings intolerable delays and risks of privacy leakage. Fortunately, with the continuous development of edge computing technologies, MEC [4] allows users to offload tasks to edge servers with sufficient computing resources for processing, which can greatly reduce transmission delay [5]. Therefore, offloading latency-sensitive and computation-intensive tasks to nearby MEC servers becomes a better choice.

Offloading a complete application to a single server is the most straightforward way, however, such offloading may not meet the QoS of these applications. Fortunately, most applications are composed of multiple inter-dependent functions or subtasks, which can be offloaded to different edge servers [6, 7]. There are many studies on scheduling dependent tasks to satisfy deadline requirements and reduce execution cost [8–10]. However, in reality, the servers can only handle relevant tasks if the corresponding service is cached. For example, when dealing with interactive online gaming tasks, the server needs to store some non-trivial data in advance. The task dependency and service caching should be considered when offloading tasks. Otherwise, the task will not be executed successfully. Thus, some works jointly consider service caching and task offloading for MEC [11–13].

In reality, a service on a server is not always requested. Typically, the arrival of tasks exhibits a distribution over time. This means that some services cached on a server will be idle most of the time. Typically, in addition to the energy to serve users, servers also need to consume a large amount of energy to maintain idle services [14, 15]. All the services in DCs remain open even at night and early morning, ready for the arrivals of users. Only 5 to 10 percent of the servers are used during these times, however, an idle server consumes 60 to 80 percent energy of that of a busy server [16]. Therefore, there are many studies focus on improving energy efficiency by reducing wasted energy.

Meanwhile, the resources of edge servers are more limited than remote cloud and edge servers generally run for a long time, which makes these idle services continue to consume energy and occupy the limited cache space, which causes a waste of resources [17]. For example, services on disk need to be loaded into memory to run, whereas server memory is limited and precious. When the number of idle services is large enough, these idle services will incur a non-negligible additional cost. In this paper, we take the state management of services into account in the service caching mechanism, which helps us manage edge servers in a finer-grained way to reduce overall cost.

It is worth noting that the resources of MEC are not unlimited, therefore, we should make effective use of these resources. However, task offloading with service caching and state management has the following challenges: (1) Service caching and task offloading are highly coupled, it is challenging to cache services within limited edge server resources to improve performance. (2) As mentioned before, letting idle services occupy memory is a waste of resources, but placing all idle services on disk is not desirable. Because the tasks come continuously, the currently idle service may be requested

*The corresponding author is Sheng Zhang (sheng@nju.edu.cn). This work was supported in part by NSFC (61872175, 61832008), the Fundamental Research Funds for the Central Universities (2022300297), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

at the next moment, and loading the service from the disk into the memory also leads to an extra delay. Although there has been a lot of work studying stateful service placement and migration, few are related to task offloading. Therefore, it is necessary yet challenging to determine the service states at each moment in long-term task offloading..

Our main contributions are summarized as follows:

- We define the Cost-efficient Delay-bounded dependent task Offloading with Service management problem (CDOS). *To our best knowledge, this is the first time such problem is studied.*
- We propose DCSO, an online algorithm, which can efficiently cache services and manage the state of service in an online manner. DCSO leverages the Lyapunov method to transform the original problem into a series of single-time frame minimization problems, each of which is solved by using the Gibbs sampling. Meanwhile, DCSO achieves a 2 competitive ratio in state management.
- We perform trace-based simulations and show that the proposed DCSO algorithm significantly reduces the total cost compared to baseline algorithms.

II. Model and Problem Definition

A. System Model

As we can see from Fig. 1, time is divided into frames, where a frame contains a control sub-frame and an offloading sub-frame. We use $\mathbb{T}=\{1,2,...,\tau\}$ to denote the total time frames. In the control sub-frame, control information is exchanged among edge servers and mobile devices to determine the offloading schedule and services′ state and tasks are processed during offloading sub-frame. We use π_c and π to denote the duration of a control sub-frame and the entire time frame, respectively. Edge servers in the MEC system are denoted as $\mathbb{E}=\{e_1,e_2,...,e_m\}$ and there are N users need to offload jobs to MEC for processing. The set of services is denoted as $\mathbb{A}=\{a_1,...,a_s\}$.

Each edge server can cache multiple services as long as there are available resources. For convenience, this paper considers the CPU and memory resource only. We use W_i to represent the computing capacity of edge server e_i. The state management of services in a real environment is very complex and difficult to model. To simplify the analysis, we assume that the server is at maximum CPU speed when processing tasks, and at minimum CPU speed when the server is idle [13]. We assume that the services that have been cached on the server have three states, namely 'work′, 'ready′ and 'sleep′. Service a_s is in state 'work′ when a task is being processed by the a_s, and we use $c^e_{s,j}$ to denote the cost when service a_s execute one unit workload on edge server j. Service a_s is in state 'ready′, if the service is loaded into memory of the edge server and can execute the incoming tasks immediately and we use c^p_s to represent its unit cost. Service a_s is in state 'sleep′, if the service is cached by the edge server but it needs to be activated before processing the tasks. Like [18], we use c^w_s to denote the activation cost of service a_s, which mainly comes from

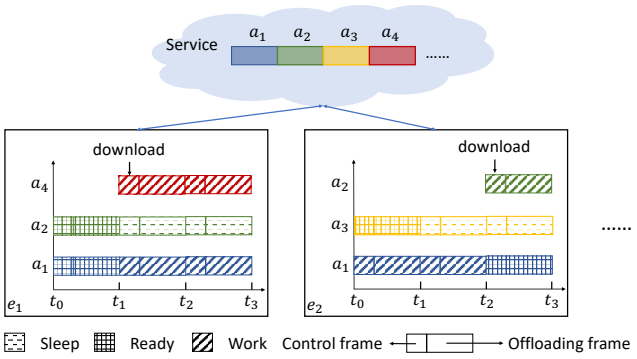


Fig. 1. Service Caching and State Management.

the maintenance cost of the edge system. Denote $I^j_s(t)$ as an indictor, $I^j_s(t)$=1 if service a_s is cached by edge server e_j in time t, $I^j_s(t)$=0 otherwise. Denote $U^j_s(t)$ as an indictor, $U^j_s(t)$=1 if service a_s is cached and is in state 'sleep′ in time t, $U^j_s(t)$=0 otherwise.

In this paper, each job is composed of multiple dependent tasks and a job from user k is represented as a directed acyclic graph (DAG) $J_k(t)$, where t indicates the time frame. We use $\mathbb{V}_k(t)$ to represent the set of tasks in $J_k(t)$ and $v^i_k(t)$ to represent the i-th task in $\mathbb{V}_k(t)$. $b^i_k(t)$ represents the workload of task $v^i_k(t)$ in job $J_k(t)$ in the time frame t. For consistency if user does not need to offload job in frame t, the workload of all tasks in $J_k(t)$ is zero. A directed edge (i,j) in the DAG of $J_k(t)$ indicates that task $v^i_k(t)$ must be processed before task $v^j_k(t)$, and we use $\alpha^{ij}_k(t)$ to represent the amount of data exchange between task $v^i_k(t)$ and $v^j_k(t)$. For ease of presentation, we use $R^i_k(t)$ to denote the set of precursor tasks of v^i_k in job $J_k(t)$. Each task exactly maps to a service that needs to be cached on servers to process and we use $\phi^i_k(t)$ to denote the service that is used to process task $v^i_k(t)$.

If task $v^i_k(t)$ is offloaded to edge server e_j without the service to execute it, the corresponding service must be downloaded from the remote-cloud in the control-frame of time frame t. In Fig. 1, service a_4 is downloaded in the control frame in t_1 by edge e_1. Denote $x^{i,j}_k(t)$ as an indictor, $x^{i,j}_k(t)$=1 if task $v^i_k(t)$ is offloaded to e_j in time t, $x^{i,j}_k(t)$=0 otherwise. We use $c^r_{s,j}$ to represent the cost when downloading service a_s on server e_j and we assume that all services can be downloaded and cached during the control sub-frame. Thus, the download cost in time frame t can be denoted as

$$C_D(t)=\sum_{k=1}^{N}\sum_{i=1}^{|\mathbb{V}_k(t)|}\sum_{j=1}^{M}x^{i,j}_k(t)(1-I^j_{\phi^i_k(t)}(t-1))c^r_{\phi^i_k(t),j}, \quad (1)$$

where $x^{i,j}_k(t)(1-I^j_{\phi^i_k(t)}(t-1))=1$ represents that task $v^k_i(t)$ is offloaded to server e_j and need to download service $\phi^i_k(t)$ for processing. We use $d^e_{s,j}$ and $c^e_{s,j}$ to represent the time and cost when execute one unit workload of service a_s on edge server e_j, respectively. Therefore, the execution cost $C_E(t)$ can be denoted as $C_E(t)=\sum_{k=1}^{N}\sum_{i=1}^{|\mathbb{V}_k(t)|}\sum_{j=1}^{M}x^{i,j}_k(t)b^i_k(t)c^e_{ij}$. Similarly, we use d^t_{uv} and c^t_{uv} to represent the time and cost when transferring one unit data from edge server e_u to e_v,

respectively. The transmission cost $C_T(t)$ in time frame t is as follows:

$$C_T(t)=\sum_{k=1}^{N}\sum_{i=1}^{|\mathbb{V}_k(t)|}\sum_{\substack{h\in R_k^i(t),\\ x_k^{h,p}(t)=1}}\left(\sum_{j=0}^{M}x_k^{i,j}(t)\alpha_k^{h,i}(t)c_{pj}^t\right), \quad (2)$$

where the third summation represents the amount of data transferred from precursor tasks to task $v_k^i(t)$.

We use colocation cost to represent the cost when service is in 'ready′ state. For services that cached on the edge server but have not processed tasks throughout the time frame, its idle time is π. For services that is in the 'work′ state at the beginning of the time frame, its idle time is $(\pi-b_k^i(t)d^e_{\phi_k^i(t),j})$. For services that need to download from the cloud, its idle time is $\pi-\pi_c-b_k^i(t)d^e_{\phi_k^i(t),j}$. Thus, the colocation cost $C_P(t)$ is denoted as

$$\begin{aligned}&\sum_{k=1}^{N}\sum_{i=1}^{|\mathbb{V}_k(t)|}\sum_{j=1}^{M}c^p_{\phi_k^i(t)}\{\pi(1-U^j_{\phi_k^i(t)}(t))(1-x_k^{i,j}(t))I^j_{\phi_k^i(t)}(t)\\&+(\pi-b_k^i(t)d^e_{\phi_k^i(t),j})x_k^{i,j}(t)(1-U^j_{\phi_k^i(t)}(t-1))\\&+(\pi-\pi_c-b_k^i(t)d^e_{\phi_k^i(t),j})x_k^{i,j}(t)(U^j_{\phi_k^i(t)}(t-1)+1-I^j_{\phi_k^i(t)}(t-1))\}.\end{aligned} \quad (3)$$

The activation cost $C_W(t)$ is denoted as

$$C_W(t)=\sum_{k=1}^{N}\sum_{i=1}^{|\mathbb{V}_k(t)|}\sum_{j=1}^{M}x_k^{i,j}(t)U^j_{\phi_k^i(t)}(t)c^w_{\phi_k^i(k)}. \quad (4)$$

Therefore, the total cost in time frame t is $C(t)=C_E(t)+C_D(t)+C_P(t)+C_T(t)+C_W(t)$. Meanwhile, we denote $D_E(k,t)$ as the execution latency of task $J_k(t)$ in time frame t, and

$$D_E(k,t)=\sum_{i=1}^{|\mathbb{V}_k(t)|}\sum_{j=1}^{M}x_k^{i,j}(t)b_k^i(t)d^e_{\phi_k^i(t),j}. \quad (5)$$

The accumulated latency after executing task $v_k^i(t)$ is denoted as $D(i,k,t)$, and the accumulated transmission time is determined as

$$\max_{\substack{h\in R_k^i(t),\\ x_k^{h,p}(t)=1}}\left(D(h,k,t)+\sum_{j=0}^{M}x_k^{i,j}(t)\alpha_k^{h,i}d_{pj}^t\right), \quad (6)$$

where $\alpha_k^{h,i}(t)$ denotes the amount of data exchange between task $v_k^h(t)$ and $v_k^i(t)$ in job $J_k(t)$. So the accumulated completion time from the first time to the task $|\mathbb{V}_k(t)|$ is as follows:

$$\begin{aligned}&D(|\mathbb{V}_k(t),k,t|)\\&=D_E(k,t)+\max_{\substack{h\in R_k^i(t),\\ x_k^{h,p}(t)=1}}\left(D(h,k,t)+\sum_{j=0}^{M}x_k^{i,j}(t)\alpha_k^{h,i}d_{pj}^t\right).\end{aligned} \quad (7)$$

Alg. 1: Dynamic Caching with State Optimization (DCSO) alg. for **P1**

Input: $q(0)\leftarrow 0$;
Output: service caching policy $\{\mathbf{\Lambda}^1,\mathbf{\Lambda}^2,...,\mathbf{\Lambda}^\tau\}$ and task offloading policy $\{\mathbf{X}^1,\mathbf{X}^2,...,\mathbf{X}^\tau\}$;
for *t=0 to* $\tau-1$ **do**
  Receive user tasks from environment;
  for *service* a_s *cached on server* e_j **do**
    if $U_s^j(t)=0$ **then**
      if $U_s^j(t)=0$ *lasts* c_s^w *time frames:* **then**
        $U_s^j(t)=1$.
  Choose $\{x_{ij},U_{ij}\}$ by solving **P2** using GS alg.;
  $q(t+1)=\max\{q(t)+D(|\mathbb{V}_k(t)|,k,t)-Q,0\}$;

B. Problem Formulation

The Cost-efficient Delay-bounded dependent task Offloading with Service management problem (CDOS) is as follows:

$$\textbf{(P1)}\quad \min\lim_{\tau\to\infty}\frac{1}{\tau}\sum_{t=1}^{\tau}C(t) \quad (8a)$$

$$s.t.\quad x_k^{i,j}(t)\le 1-U^j_{\phi_k^i(t)}(t) \qquad \forall k\in\mathbb{J},i\in\mathbb{V}_k,j\in\mathbb{E},t\in\mathbb{T}, \quad (8b)$$

$$U^j_{\phi_k^i(t)}(t)\le I^j_{\phi_k^i(t)}(t) \qquad \forall i\in\mathbb{A},j\in\mathbb{E},t\in\mathbb{T}, \quad (8c)$$

$$\sum_{i=1}^{N}I^j_{\phi_k^i(t)}(t)\le 1 \qquad \forall j\in\mathbb{E},t\in\mathbb{T}, \quad (8d)$$

$$\sum_{i=0}^{s}I^j_{\phi_k^i(t)}(t)w_{\phi_k^i(t)}\le W_j \qquad \forall j\in\mathbb{E},t\in\mathbb{T}, \quad (8e)$$

$$\lim_{\tau\to\infty}\frac{1}{\tau}\sum_{t=1}^{\tau}D(|\mathbb{V}_k(t)|,k,t)\le Q \qquad \forall k\in\mathbb{J},j\in\mathbb{E}. \quad (8f)$$

Eq. (8b) represents the service caching constraint. Eq. (8d) indicates that at most one instance of each service can be cached on the server. Eq. (8e) represents the resource constraint. Eq. (8f) is the long-term delay constraint for the tasks of users, which couples the offloading decision both spatially and temporally.

III. Algorithm

A. Lyapunov Method

In the online scenario, the decisions made at the present time frame will have an impact on the future offloading decisions, therefore, we cannot directly solve **P1**. With Lyapunov method, we can focus on solving a series of minimization problems in each time frame. First, we use a virtual queue to measure the exceeded latency by history. Specially, assuming $q(0)=0$, we have: $q(t+1)=max\{q(t)+D(|\mathbb{V}_k(t)|,k,t)-Q,0\}$ where $q(t)$ indicates the gap between current delay from the constraint. The congestion of the virtual queue is represented by the Lyapunov function $L(q(t))=\frac{1}{2}q^2(t)$. A small $L(q(t))$

implies that the virtual queue is stable and not congested. We introduce the one-frame Lyapunov drift to stabilize the queue: $\Delta(q(t))=\mathbb{E}[L(q(t+1))-L(q(t))|q(t)]$

Theorem 1. *For $q(t)$ in all frames, the following stable statement always holds: $\Delta(q(t))\leq B+q(t)\mathbb{E}[(D(|\mathbb{V}_k(t)|,k,t)-Q)|q(t)]$, where $B=\frac{1}{2}(D_{max}(|\mathbb{V}_k(t)|,k,t)-Q)^2$ is a constant value, and $D_{max}(|\mathbb{V}_k(t)|,k,t)=max_{t\in\tau}D(|\mathbb{V}_k(t)|,k,t)$ represents the biggest average latency in all frames.*

Proof.

$$\Delta(q(t))=\frac{1}{2}\mathbb{E}[q^2(t+1)-q^2(t)|q(t)] \tag{9}$$

$$\leq\frac{1}{2}\mathbb{E}[(q(t)+D(|\mathbb{V}_k(t)|,k,t)-Q)^2-q^2(t)|q(t)] \tag{10}$$

$$=\frac{1}{2}(D(|\mathbb{V}_k(t)|,k,t)-Q)^2+q(t)\mathbb{E}[(D(|\mathbb{V}_k(t)|,k,t)-Q)|q(t)] \tag{11}$$

$$\leq B+q(t)\mathbb{E}[(D(|\mathbb{V}_k(t)|,k,t)-Q)|q(t)]. \tag{12}$$

The inequality (10) comes from $(q(t)+D(|\mathbb{V}_k(t)|,k,t)-Q)\geq max\{q(t)+D(|\mathbb{V}_k(t)|,k,t)-Q,0\}$. □

Given the MEC state information in time frame t, $\Delta(q(t))$ represents the expected change in Lyapunov function. The smaller $\Delta(q(t))$ is, the more stable the virtual queue is. We focus on finding the optimal offloading strategies for all DAG tasks and service caching decisions for edge servers, taking into account the state management of services. Therefore, we need to minimize the upper bound of the drift-plus-cost expression in each time frame:

$$\Delta(q(t))+V\cdot\mathbb{E}[C(t)|q(t)] \tag{13}$$
$$\leq B+q(t)\mathbb{E}[(D(|\mathbb{V}_k(t)|,k,t)-Q)|q(t)]+V\cdot\mathbb{E}[C(t)|q(t)].$$

V is a positive parameter, used to adjust the tradeoff between cost and latency. A minimized upper bound of the drift-plus-penalty term can be obtained by the drift-plus-penalty algorithm [19] in Lyapunov optimization. Thus, the new minimization problem is as follows:

$$\textbf{(P2)} \quad \min_{x,I,U} \quad (V\cdot C(t)+q(t)D(|\mathbb{V}_k(t)|,k,t)) \tag{14}$$

$$s.t. \quad \text{(8b)}-\text{(8e)}. \tag{15}$$

In each time frame, task processing and data transmission will cause delays. We consider the average delay of this part by considering the addition term $q(t)D(|\mathbb{V}_k(t)|,k,t)$. Minimizing the average latency is more important when $q(t)$ is big. As a consequence, our algorithm follows "Use more servers and remote cloud if latency constraint is violated" philosophy, and the latency is maintained, thereby enabling online offloading decisions. In this way, the algorithm is completed as long as **P2** is solved, which will be discussed in Section III-B.

Each service cached by the edge node has three states, for a service in ready state, we need to decide when and whether to turn it into sleep state. The state of services cached on the edge server is determined by a wait and sleep way (Lines 5−9 in alg. 1). The method is simple and straightforward, if a service is in ready state for c_s^w time frame, we then turn it into sleep state. If the service turn to work state quickly (within c_s^w frames), this state management is optimal. If the service turn to work state after a long time frames (longer than c_s^w), then this state management spends $2c_s^w$ cost, while the optimal algorithm solution spends only c_s^w cost. Because the optimal solution turns the service into sleep state once it finishes the task. Thus, the state management in alg. 1 is 2-competitive.

Alg. 2: Gibbs Sampling based (GS) alg. for **P2**

1 Output: Caching policy $\mathbf{\Lambda}$ and offloading policy $\mathbf{X}$;
2 Initiation: $\mathbf{\Lambda^0}\leftarrow 0$;
3 **for** *iteration* i=1,2,...,N **do**
4 Randomly choose an edge server $r\leq M$ and a caching decision $\lambda'_{\mathbf{r}}$;
5 **if** $\lambda'_{\mathbf{r}}$ *is feasible* **then**
6 Given caching policy $\mathbf{\Lambda^{i-1}}$, compute the offloading decision $\mathbf{X}$ and cost C;
7 Given caching policy ($\mathbf{\Lambda^{i-1}}\bigcup\lambda'_r$), compute the optimal offloading decision $\mathbf{X'}$ and cost C';
8 Let $\lambda^{\mathbf{i}}_{\mathbf{r}}=\lambda'_{\mathbf{r}}$ with the probability $p=\frac{1}{1+e^{(C'-C)/\omega}}$ and $\lambda^{\mathbf{i}}_{\mathbf{r}}=\lambda^{\mathbf{i-1}}_{\mathbf{r}}$ with probability $1-p$;
9 Return $\mathbf{X^N}$,$\mathbf{\Lambda^N}$.

B. Optimization Algorithm for Single Time Frame

We focus on the cost minimization problem **P2** in each time frame. It is well known that the joint service caching and task offloading is already NP-complete without considering service state selection [13]. We present algorithm 2 based on Gibbs sampling. Algorithm 2 iteratively determines the optimal caching and offloading decisions at the control frame. In each iteration, an edge server r ($r\leq M$) changes caching decision from $\lambda_{\mathbf{r}}$ to $\lambda'_{\mathbf{r}}$ virtually while keeping other caching decisions unchanged (Step 5). GS will check the feasibility of the changed service caching decision (Step 6). If the changed decision is feasibility, we then can get the corresponding objective value C. Assume that when caching decision of edge server r changes from $\lambda_{\mathbf{r}}$ to $\lambda'_{\mathbf{r}}$, the objective cost varies from C to C'. Then, we get the relationship between the conditional probability distribution of service caching strategies and the objective cost: the caching decision of edge server r changes from $\lambda_{\mathbf{r}}$ to $\lambda'_{\mathbf{r}}$ with probability $p=\frac{1}{1+e^{(C'-C)/\omega}}$ ($\omega>0$) and maintains $\lambda_{\mathbf{r}}$ with probability $1-p$ (Step 9).

It is well known that decisions made based on current information in combinatorial optimization tend to fall into local optima. Thus, Gibbs sampling uses a smooth parameter $\omega>0$ to control exploration and exploitation. Gibbs Sampling has a good property that when ω becomes smaller, the algorithm can converge to the global optimal solution with an increasing probability. And alg. DCSO converges to the optimal result in global with probability 1 when $\omega\rightarrow 0$.

C. Algorithm Analysis

The performance of DCSO is analyzed in this subsection.

Theorem 2. *Using alg. DCSO, the average delay is $O(\frac{1}{V})$, and the average queue size is $O(V)$.*

Proof.

Lemma 1. *For any $\delta \geq 0$, there exists a stationary and randomized policy Π for **P1**, which decides independent of the current queue backlogs $q(t)$, such that:*

$$E[D(|\mathbb{V}_k(t)|,k,t)^{\Pi} - Q] \leq \delta. \tag{16}$$

The proof can be obtained by Theorem 4.5 in [20]. By applying lemma 1, we have:

$$\begin{aligned}
&\Delta(q(t)) + V \cdot E[C(t)^{\Pi} | q(t)] \\
&\leq B + q(t) E[D(|\mathbb{V}_k(t)|,k,t)^{\Pi} - Q | q(t)] + \\
&V \cdot E[D(|\mathbb{V}_k(t)|,k,t)^{\Pi} | q(t)] \\
&\leq B + \delta q(t) + V \cdot (C(t)^{opt}) + \delta). \qquad (17)
\end{aligned}$$

Let $\delta = 0$, summing the right inequality over $\tau - 1$ time frames and then dividing the result by τ, we have:

$$\frac{1}{\tau} E[L(q(t)) - L(q(0))] - \frac{V}{\tau} \sum_{t=0}^{\tau-1} E[C(t)^{\Pi}] \leq B + V \cdot C(t)^{opt}. \tag{18}$$

Because $L(q(t)) \geq 0$ and $L(q(0)) = 0$ in our scenair, we have: $\lim_{\tau \to \infty} \frac{1}{\tau} \sum_{t=0}^{\tau-1} E[C(t)] \geq C(\tau)^{opt} - \frac{B}{V}$.

After that, we assume there are values $\epsilon > 0$ and $\Psi(\epsilon)$ and an decision $C(t)^{\Gamma}$ that satisfies: $E[C(t)^{\Gamma}] = \Psi(\epsilon)$, $E[D(|\mathbb{V}_k(t)|,k,t)^{\Gamma} - Q] \leq -\epsilon$. Plugging above into inequality (13): $\Delta(q(t)) + V E[C(t)^{\Gamma}] \leq B + V\Psi(\epsilon) - \epsilon q(t)$.

Adding up the above formulas by time frames, we have:

$$\begin{aligned}
\frac{1}{\tau} \sum_{t=0}^{\tau-1} E[q(t)] &\leq \frac{B + V\left(\Psi(\epsilon) - \frac{1}{\tau} \sum_{t=0}^{\tau-1} E[C(t)^{\Gamma}]\right)}{\epsilon} \\
&\leq \frac{B}{\epsilon} + \frac{V}{\epsilon}\left(C^{max} - C^{opt}\right). \qquad (19)
\end{aligned}$$

Considering $\sum_{t=0}^{\tau-1} \geq \sum_{t=0}^{\tau-1} E[D(|\mathbb{V}_k(t)|,k,t)^{\Gamma} - Q]$, we have:

$$\frac{1}{\tau} \sum_{t=0}^{\tau-1} E[D(|\mathbb{V}_k(t)|,k,t)^{\Gamma}] \leq \frac{B}{\epsilon} + \frac{v}{\epsilon}(C^{max} - C^{opt}) + Q. \tag{20}$$

Taking a lim sup of Eq. (20) as $t \to \infty$ yields the delay bound. □

IV. Performance Evalution

We compare DCSO with the following baseline algorithms.

- Non-State algorithm (Non-State) [21]: Edge serves cache services according to Alg. 2. Once a service is cached on the server, it never switches to sleep state;
- Cost Priority algorithm (CP): Edge servers cache services according to caching cost regardless of long-term constraint. Once a service is idle, it switches to sleep state;
- Cloud Only algorithm (CO): This algorithm always offloads tasks to cloud for processing.

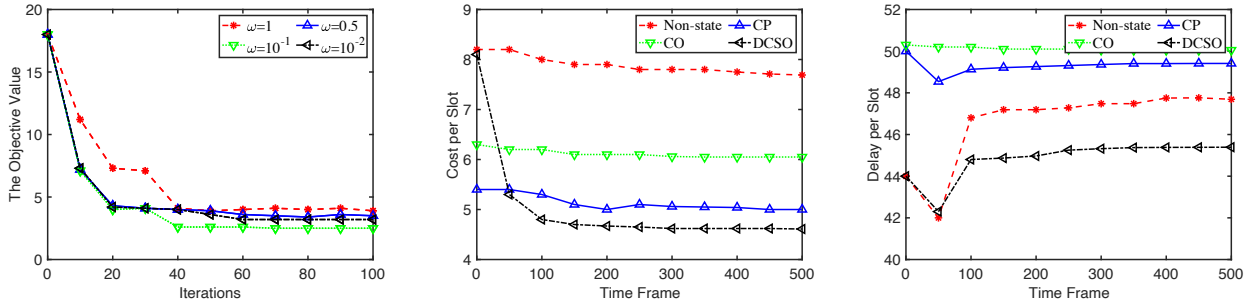
There are 10 types of service and each of 10 servers can cache 3 to 5 services simultaneously, and the remote-cloud caches all services. We perform the simulations using Alibaba′s trace of data [22]. To comply the characteristics of low latency in MEC, the transmission delay among edge servers is set to [5,100] ms and and the processing time of each task is set to [10,200] ms. The transmission delay from the edge server to the remote-cloud is set 20 times of that between two edge servers, because the distance between the edge server and the remote-cloud is very long and data needs to be transmitted through the WAN. Since the remote-cloud usually provides more resources than edge servers, the processing time in the remote-cloud is set 0.75 times of that in MEC. Note that once a task is uploaded to the remote-cloud for processing, subsequent tasks will be executed on the cloud because the latency and the cost of transmission is too high. Costs are measured in energy consumption. We set the transmission cost as 0.935 J/KB and the caching cost as 1.5 J [23]. We set the processing cost as [2,4] J/KB of each edge server and 5 J/KB of the cloud. The ready cost is set to be 0.8 J and the activation cost is set to be 1.2 J.

A. Simulation Results

1) *Convergence:* Fig. 2(a) shows the convergence during the execution of DCSO.. The smaller ω is, the faster convergence speed is. When $\omega = 10^{-2}$, our algorithm DCSO has converged within 40 iterations. However, it is incorrect to reduce ω blindly, which will not make the algorithm converge to the optimal solution. In our experiment, when the value of ω is 10^{-1}, a good compromise can be achieved between the solution and the speed of convergence.

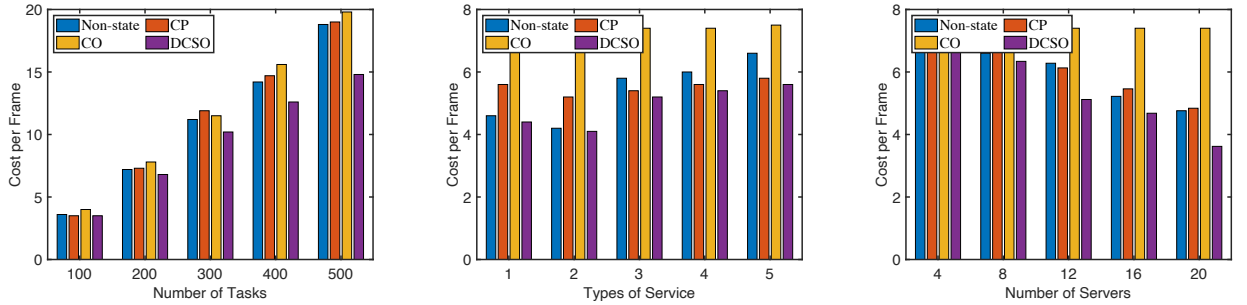
2) *The impact of different parameters:* From Fig. 2(b) and Fig. 2(c), we can see that DCSO outperforms the other baseline algorithms. CO alg. has the worst performance because it always offloads tasks to the remote-cloud, resulting in a large transmission delay. And as expected, the cost of Non-state alg. is still large, this is because idle services consume too much energy. The cost of CP alg. is not the minimal, because the service will enter sleep mode once idle, and the activation operation will also consume a lot of energy.

3) *Performance comparison:* In Fig. 3(a), as the number of tasks increases, each time requires more resources to process tasks, thus increases the cost. It is worth noting that when the type of service changes from 1 to 2, we set the cost and workload of the second service 0.7 times that of the first service. So there is some reduction in the cost of all algorithms (Fig. 3(b)). Fig. 3(c) shows the impact of number of servers. There is little change in the performance of the CO algorithm as it uploads all tasks to the cloud for processing. When the number of servers is small, in order to handle different types of tasks, the server will frequently update the cached service, which brings a lot of costs. Therefore, when the number of servers increases, the processing of newly arrived tasks can reuse preciously cached services, thus reducing a lot of costs. From Fig. 3(b) and Fig. 3(c), since CP alg. set idle services to sleep state, when there are few types of services, the activation



(a) Convergence of DCSO. (b) Cost per frame. (c) Delay per frame.

Fig. 2. The impact of different parameters for DCSO.



(a) Impact of number of tasks. (b) Impact of types of services. (c) Impact of number of servers.

Fig. 3. The performance of different algorithms

operation will be triggered frequently, which will increase a lot of costs. And since Non-state alg. does not consider state management, when there are more idle services (more service types and fewer tasks), more costs will be incured.

V. Conclusion

Because the resources of edge system are limited, caching appropriate services in MEC and offloading tasks to appropriate servers is crucial for optimizing quality of experience and improving resource utilization. We consider the dependent tasks offloading and scheduling with service caching and state management in MEC. We propose DCSO, an online algorithm that considers dynamic service caching and state management to minimize long-term cost in MEC systems. Experiments demonstrate our algorithm outperforms other baselines.

References

[1] J. Xiong, E.-L. Hsiang, Z. He, T. Zhan, and S.-T. Wu, "Augmented reality and virtual reality displays: emerging technologies and future perspectives," *Light: Science & Applications*, vol. 10, no. 1, pp. 1–30, 2021.

[2] B. Sonkoly, J. Czentye, M. Szalay, B. Németh, and L. Toka, "Survey on placement methods in the edge and beyond," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2590–2629, 2021.

[3] M. M. Rathore, H. Son, A. Ahmad, and A. Paul, "Real-time video processing for traffic control in smart city using hadoop ecosystem with gpus," *Soft Computing*, vol. 22, no. 5, pp. 1533–1544, 2018.

[4] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE communications surveys & tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[6] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2777–2792, 2021.

[7] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3056–3069, 2017.

[8] Y. Fan, L. Zhai, and H. Wang, "Cost-efficient dependent task offloading for multiusers," *IEEE Access*, vol. 7, pp. 115 843–115 856, 2019.

[9] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 37–45.

[10] X. Qin, B. Li, and L. Ying, "Distributed threshold-based offloading for large-scale mobile cloud computing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[11] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2076–2085.

[12] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1997–2006.

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

[14] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu *et al.*, "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems," *Computing*, vol. 98, no. 7, pp. 751–774, 2016.

[15] S. Jin, X. Qie, W. Zhao, W. Yue, and Y. Takahashi, "A clustered virtual machine allocation strategy based on a sleep-mode with wake-up threshold in a cloud environment," *Annals of Operations Research*, vol. 293, no. 1, pp. 193–212, 2020.

[16] L. Duan, D. Zhan, and J. Hohnerlein, "Optimizing cloud data center energy efficiency via dynamic prediction of cpu idle intervals," in *2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 2015, pp. 985–988.

[17] Z. Xu, Y. Qin, P. Zhou, J. C. Lui, W. Liang, Q. Xia, W. Xu, and G. Wu, "To cache or not to cache: Stable service caching in mobile edge-clouds of a service market," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 421–431.

[18] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 468–476.

[19] F. Liu, P. Shu, and J. C. Lui, "Appatp: An energy conserving adaptive mobile-cloud transmission protocol," *IEEE transactions on computers*, vol. 64, no. 11, pp. 3051–3063, 2015.

[20] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[21] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[22] "Alibaba Inc. 2018. Evolution of Alibaba Large-Scale Colocation Technology." https://www.alibabacloud.com/blog/evolution-of-alibaba-large-scale-colocation-technology_594172.

[23] B. Flipsen, J. Geraedts, A. Reinders, C. Bakker, I. Dafnomilis, and A. Gudadhe, "Environmental sizing of smartphone batteries," in *2012 Electronics Goes Green 2012+*. IEEE, 2012, pp. 1–9.