Caching Assisted Correlated Task Offloading for IoT Devices in Mobile Edge Computing

Chaogang Tang*, Chunsheng Zhu†‡, Huaming Wu§, Chunyan Liu¶, Joel J. P. C. Rodrigues‖**
*School of Computer Science and Technology, China University of Mining and Technology, 221116, Xuzhou, China
†Institute of Future Networks, Southern University of Science and Technology, 518055, Shenzhen, Guangdong, China
‡PCL Research Center of Networks and Communications, Peng Cheng Laboratory, 518055, Shenzhen, Guangdong, China
§ The Center for Applied Mathematics, Tianjin University, 300072, Tianjin, China
¶ College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, 211106, Nanjing, China
‖ Federal University of Piauí, Teresina - Pi, Brazil
** Instituto de Telecomunicações, Portugal
cgtang@cumt.edu.cn, chunsheng.tom.zhu@gmail.com, whming@tju.edu.cn, lcy_cs@nuaa.edu.cn, joeljr@ieee.org

GLOBECOM 2021 - 2021 IEEE Global Communications Conference | 978-1-7281-8104-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/GLOBECOM46510.2021.9685828

***Abstract*—The fast-growing Internet of Thing (IoT) has generated a vast number of tasks which need to be performed efficiently. Owing to the drawback of the sensor-to-cloud computing paradigm in IoT, mobile edge computing (MEC) has become a hot topic recently. Against this backdrop, we focus on the offloading of tasks characterized by intrinsic correlations in this paper, which have not been considered in most of existing works. For the sequential arrival of such correlated tasks, the future workload can be efficiently reduced by caching the current computational result. Specifically, we resort to the Lyapunov optimization to handle the long-term constraint on energy consumption. Simulation results reveal that our approach is superior to other approaches in the optimization of response latency and energy consumption.**

***Index Terms*—Caching, correlated task offloading, MEC, response latency, energy consumption**

I. Introduction

The fast-growing development of Internet of Things (IoT) has given rise to a ubiquitous network connecting IoT devices [1], [2]. Considering the restricted computational capabilities and energy reserve of these IoT devices, generated tasks can be processed in the sensor-to-cloud computing paradigm, where tasks are offloaded and executed in a remote cloud center. Such a computing paradigm exploits the unconstrained computing resources at the cloud, but it might incur long response latency because of data transmission in the backbone network. Therefore, it is not suitable for latency sensitive tasks. Requirements such as energy reduction and response latency optimization have stimulated the emergence of compromised computing paradigms to mitigate the pressure of these IoT devices [3], [4]. For instance, mobile edge computing (MEC) has become a hot topic in recently years, for its cloud-similar computing capabilities with less response latency. Specifically, MEC strives to deploy computing facilities at the network edge such as macro base station (MBS), thus enabling the shift of computational workload from the cloud center to the network edge. MEC makes task perform in close proximity to IoT devices where tasks are generated, so the response latency can be greatly reduced as expected.

On another hand, a huge number of tasks need to be offloaded, which raises the possibility of repetitive offloading for the same task. Furthermore, such repetitive task offloading not only wastes unnecessary energy consumption but also incurs long response latency. Accordingly, caching strategy can be introduced for avoiding such concerns [5]. Generally, caching the most frequently requested tasks for IoT devices in MEC can dramatically shorten the response latency, for the reason that the execution results can be directly returned without repeated execution.

For instance, service caching benefits the resource-limited edge server when tasks are offloaded. Authors in [6] investigate this issue in depth by optimizing task offloading together with service caching in MEC, and they propose an efficient algorithm to tackle service heterogeneity, system dynamics, and other key challenges in MEC. Authors in [7] aims to jointly optimize task caching and offloading in MEC, from the computing and storing resource constrained perspective. The problem is modeled as a mixed integer programming that is solved by an alternating iterative algorithm.

Authors in [8] try to cache the computational result in a proactive way in fog computing. They model it as a long-term weighted-sum energy minimization problem, and solve it by a sliding-window based online algorithm. Authors in [9] leverage task caching to enhance the computing capabilities of MEC, by jointly optimizing caching, computation, and communication resources in MEC. The offloading efficiency usually has a great effect upon the quality of experience (QoE) [10], [11], and service caching can help improve the offloading efficiency. Authors in [12] strive to optimize both service caching and task offloading in MEC from the viewpoint of QoE. They design a utility function based on QoE and formulate the optimization as an integer nonlinear programming problem. Authors in [13] endeavor to achieve intelligent task caching in the edge-cloud environment. By doing this, they can generalize the task caching strategies and ignore some unrealistic assumptions in existing works such as knowing the pattern of user task requests. To this end, a multi-

978-1-7281-8104-2/21/$31.00 ©2021 IEEE

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on February 04,2022 at 01:09:13 UTC from IEEE Xplore. Restrictions apply.

armed bandit algorithm is adopted to help learn the pattern of tasks and dynamically adjust the caching strategy.

However, we notice that most of the aforementioned works hardly ever mine the intrinsic correlation of the task arrivals especially for the same IoT device. The computational result for the current task can actually benefit the following task if a certain correlation exists between them. It shall be noted that the two tasks are not necessarily the same. For instance, take matrix-vector multiplication for example to motivate our work as below. Suppose that a task t_1 is offloaded and intended for calculation of $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, where $\boldsymbol{A}$ is a matrix, and $\boldsymbol{x}$ is a vector as the task-input data. The next task t_2 arrives for calculation of $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{z}$, where $\boldsymbol{B} = \boldsymbol{A} + \boldsymbol{\mathcal{A}}$, $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{\eta}$, and $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\eta}$ denote a sparse matrix and vector, respectively. It is clear that caching the calculation result of t_1 is very beneficial to t_2, owing to the computational cost reduction. Such correlated tasks can be easily observed from one IoT device or functionally similar devices.

In this paper, we pay attention to such tasks which are characterized by temporarily correlated relationship, and we aim to cache the current computational result to assist the computation of future tasks. Specifically, we strive to optimize the average response latency along an infinite time horizon for the one-device one-server MEC system. In the meanwhile, a long-term energy constraint is leveraged to ensure the stability of MEC system. Specifically, the Lyapunov optimization is utilized to solve the slot spanned energy constraint such that a long-term constraint can be converted into per-slot energy constraints. On this basis, we propose an online algorithm for the minimization of the average response latency in MEC.

The rest of the paper is organized as below. Section II presents a one-device one-server system model. In Section III, we propose a Lyapunov based online algorithm to make caching decisions for the correlated tasks. Simulation results are reported and discussed in Section IV, and the conclusion comes in Section V.

II. System Model

A system model is considered which consists of one IoT device and one MBS. The edge server with the caching capability is deployed at MBS to provision computational resources to tasks offloaded by this IoT device. Time, which consists of a set of discrete time slots, can be indexed by $\{0, 1, ..., n-1\}$ and each time slot is assumed to last τ seconds. Sequential tasks with certain correlations arrive at the edge server with one task at each time slot respectively. Furthermore, suppose that each task should be accomplished by the end of its corresponding time slot. Denote the task set by $\mathcal{A} = \{a_1, a_2, ..., a_n\}$, where a_i is the task arriving at the ith time slot. a_i is a 2-tuple of (d_i, s_i), where d_i represents the average task-input data that needs to be offloaded via the wireless channel, and s_i is the average workload described by the number of CPU cycles required for performing a_i.

As exemplified above, the computational results at the current slot can assist the computation of correlated tasks at the future slots, so caching the computational results can shorten the response latency at the edge. However, it is impractical to cache all the computational results at the edge, owing to caching costs on energy consumption and storage resources [14]. In addition, we assume that the computational results are of timeliness. Specifically, any results that have been cached more than fixed time slots are no longer useful and should be abandoned. Introduce a variable I_i as an indicator to represent whether the computational result at ith time slot is cached:

$$I_i = \begin{cases} 1 & \text{if the result at slot } i \text{ is cached;} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

A. Correlated Tasks Model

In the traditional caching enabled MEC, the computational results cached at the edge can be directly used if the same tasks are repeatedly offloaded. However, such task oriented caching in MEC ignores the intrinsic correlation of the task arrivals, while the offloading for correlated tasks is far more common than the offloading for the same tasks during a finite time-slotted horizon. Whereas, it is complicated and difficult to formally model the intrinsic relationships between two sequential tasks either from coarse granularity (e.g., function level) or fine granularity (e.g., source code level), especially considering a sharp upsurge in the type and the number of tasks generated by various IoT devices.

In view of this, we investigate the correlations among tasks in a more general way, irrespective of measurement approaches or metrics. We in this paper focus on the workload reduction by making full use of previous caching decisions. To be more specific, the real workload of a_i at time slot i because of previous caching decisions can be modeled as [8]:

$$\begin{aligned} S_i =& s_i(I_{i-1}\lambda_1 + (1 - I_{i-1})I_{i-2}\lambda_2 + \cdots \\ &+ \prod_{j=1}^{k-1}(1 - I_{i-j})I_{i-k}\lambda_k + \cdots + \prod_{j=1}^{p-1}(1 - I_{i-j})I_{i-p}\lambda_p \\ &+ \prod_{j=1}^{p}(1 - I_{i-j})) \end{aligned} \tag{2}$$

where $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_p)$ with each $\lambda_j \in (0, 1)$, $j = 1, 2, ..., p$, is a vector to represent the gradually reduced influence of the previously cached results on reducing the current computational workload. From this definition, we can observe that 1) The current workload S_i can only be affected by the last caching decision in the past, e.g., if $I_{i-1} = 1$, then $S_i = s_i\lambda_1$, no matter what values I_{i-k} for $k \geq 2$ take; 2) Any computational result that has been cached more than p time slots is no longer useful, which can effectively represent the timeliness of the caching decisions; 3) The element λ_i in $\boldsymbol{\lambda}$ should be gradually increasing with increasing i, such that the closer the distance between the current time slot and the slot over which the last result is cached, the larger the effect of caching result on the current workload.

B. Communication Model

Task offloading from IoT devices is aimed for energy saving and response latency reduction, owing to the constrained

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on February 04,2022 at 01:09:13 UTC from IEEE Xplore. Restrictions apply.

computational capabilities and energy supply at the device. Considering task caching for the latency sensitive tasks, in this paper we aim for the minimization of the response latency of the offloaded tasks while satisfying the total energy constraint at the edge. The communication and computation models are respectively described as follows.

The response latency for a task offloaded and executed at the edge usually includes offloading time, execution time, and return time. The offloading time denotes the time taken to transmit the task-input data of the task into the edge via the wireless channel. Denote by g and p the channel gain between the device and the edge, and the transmission power of the device, respectively. The offloading rate of task a_i can be given as:

$$r_i = B\log_2(1+\frac{pg}{\sigma^2}) \tag{3}$$

where B is the channel bandwidth and σ^2 is the noise power. Thus, the offloading time is given as:

$$t_i^{off} = \frac{d_i}{r_i} \tag{4}$$

The energy consumption caused by task offloading can be given as:

$$e_i^{off} = pt_i^{off} \tag{5}$$

The return time means the time taken to send the computational result back to the device. Owing to the negligible size of computational result compared to the task-input data, we ignore the return delay and related energy consumption in this paper.

C. Computation Model

After tasks are offloaded to the edge successfully, the edge server will schedule resources for them. For example, the virtual resources such as virtual machines are created and computational resources are also provided in sequence. Since tasks arrive with one in each time slot respectively, and thus we ignore the queueing time at the edge. It shall be noted that task execution at the edge not only consumes the computational resources, but also incurs energy consumption, even if the caching strategy is enabled in MEC. In the following, we will respectively investigate the execution latency and energy consumption at the edge.

Based on the above description, the execution latency for a_i at the edge is defined as:

$$t_i^{exe} = t_{init} + \frac{S_i}{f_e} \tag{6}$$

where t_{init} and f_e are the time for the virtual environment initialization and the computing capability of the edge server, respectively. Therefore, the response delay for a_i is

$$t_i^{rl} = t_i^{off} + t_i^{exe} = \frac{d_i}{r_i} + t_{init} + \frac{S_i}{f_e} \tag{7}$$

The energy consumption for performing a_i at the edge can be calculated as:

$$e_i^{exe} = \kappa\varepsilon S_i f_e^2 \tag{8}$$

where κ is the effective switched capacitance coefficient and ε is the number of cycles needed to perform one task-input bit at R.

The task has to be accomplished by the end of current time slot and the edge will decide whether to cache the computational result. Recall that computational results caching will incur additional overheads on storage and energy [8]. Given the caching decisions, the resulting energy consumption at the current slot i is expressed as:

$$e_i^c = \sum_{j=i-p}^{i} I_j\gamma \tag{9}$$

where γ is the static power consumption caused by result caching for each time slot, regardless of the workload of tasks [6]. Due to the timeliness of caching results, only the last p results and the current caching decision have an effect on the energy consumption for the current time slot. Hence, the total energy consumption for a_i performing at the caching enabled MEC can be calculated as:

$$e_i^{all} = e_i^{off} + e_i^{exe} + e_i^c = pt_i^{off} + \kappa\varepsilon S_i f_e^2 + \sum_{j=i-p}^{i} I_j\gamma \tag{10}$$

D. Problem Formulation

We strive to optimize the response latency of correlated tasks offloaded to the edge with the help of previously cached computational results. Owing to the timeliness of caching results and additional overheads caused by computational results caching, it is impractical to cache all the results.

Generally, a long-term process is required for evaluating MEC system and networks when caching is enabled for correlated tasks. Therefore, we in this paper focuses on the overall response latency optimization for MEC and the optimization problem is given as below:

$$(P1) \qquad \min_{\boldsymbol{I}} \quad \lim_{n\to\infty}\frac{1}{n}\sum_{i=0}^{n-1} t_i^{rl} \tag{11}$$

s.t.

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=0}^{n-1} e_i^{all} \le Q \tag{12}$$

$$t_i^{rl} \le \tau \quad \forall i \in \{1,...,n\} \tag{13}$$

$$e_i^{all} \le e_i^{max} \quad \forall i \in \{1,...,n\} \tag{14}$$

$$I_i \in \{0,1\} \quad \forall i \in \{i-p, i-p+1, ..., i-1\} \tag{15}$$

where the constraint (12) means the overall energy consumption across different time slots must satisfy the energy constraint Q. As assumed earlier, each task should be completed by the end of its time slot, which is specified by (13). Denote by e_i^{max} the maximal energy consumption at time slot i, and the per-slot energy consumption should not exceed the corresponding energy constraint as denoted in (14). Constraint condition (15) guarantees that the last p caching decisions are binary.

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on February 04,2022 at 01:09:13 UTC from IEEE Xplore. Restrictions apply.

Challenges. Obviously, it takes exponential time for $P1$ to obtain the optimal solution, which makes the exhaustive search prohibitively costly in terms of the execution delay. Moreover, the task information at the future time slots is required for optimally solving $P1$, which can only be accomplished offline. Whereas, it is extraordinarily difficult to make predictions on the task-input data and workload in advance. Hence, an online approach is required for addressing such challenges, which can make caching decisions without the future task information.

Algorithm 1: Caching-Assisted Correlated Task Offloading Algorithm (CCTO))

Input: $\boldsymbol{\lambda}$, Q, $\mathcal{A}$, τ, n
Output: Optimum solution to $P1$

$S = 0$;
for $i = 0$ to $n-1$ **do**
  Learn d_i and s_i from the beacon information;
  Compute S_i based on Eq.(2);
  Compute t_i^{rl} based on Eq.(7);
  Obtain I_i by optimizing:
  $B - Qq(i) + \mathbb{E}[q(i)e_i^{all} + Vt_i^{rl}|q(i)]$;
  Compute e_i^{all} based on I_i;
  if $e_i^{all} < e_i^{max}$ **then**
    $S = S + t_i^{rl}$;
    $q(i+1) = \max[q(i) - Q, 0] + e_i^{all}$;
  else
    $I_i = 0$;
    Recompute e_i^{all};
    $S = S + t_i^{rl}$;
    $q(i+1) = \max[q(i) - Q, 0] + e_i^{all}$;
  end
end
$V^* = S/n$;
Return V^* and $\boldsymbol{I}$;

III. Lyapunov-Based Online Caching Decision for Correlated Tasks

A. Preliminaries

In addition to the difficulty in obtaining task information at future time slots, it is also pretty hard to handle the energy constraint that crosses different time slots. Therefore, we apply the Lyapunov optimization technology to tackling these challenges in this section. A migration queue that indicates dynamic changes of energy consumption is used for solving the slot spanned energy constraint. In particular, let $q(0) = 0$, and the queue is given as:

$$q(i+1) = \max[q(i) - Q, 0] + e_i^{all} \tag{16}$$

where $q(i)$ is the queue backlog in time slot i. From this definition, we can see that $q(i)$ can indicate the deviation of current energy consumption from the energy constraint Q. Specifically, $q(i)$ with a larger value denotes a sharper deviation. The Lyapunov function can be defined as $\mathcal{L}(q(i)) = \frac{1}{2}q^2(i)$. The stability of this queue requires that the increment between two consecutive states should be extremely small. To efficiently control such an increment, the *Lyapunov drift* is given as $\triangle(q(i)) \triangleq \mathbb{E}[\mathcal{L}(q(i+1)) - \mathcal{L}(q(i))|q(i)]$.

Lemma 1: Assume X, Y, Z and k are non-negative real numbers and $Z = \max\{Y - k, 0\} + X$, then $Z^2 \leq X^2 + Y^2 + k^2 - 2Y(k - X)$.

Based on this Lemma [15], the upper bound of $\triangle(q(i))$ can be determined, as below:

$$\begin{aligned}
\triangle(q(i)) &= \mathbb{E}[\mathcal{L}(q(i+1)) - \mathcal{L}(q(i))|q(i)] \\
&= \frac{1}{2}\mathbb{E}[[\max[q(i) - Q, 0] + e_i^{all}]^2 - q^2(i)|q(i)] \\
&\leq \frac{1}{2}\mathbb{E}[Q^2 + (e_i^{all})^2 + 2q(i)(e_i^{all} - Q)|q(i)] \\
&= \frac{1}{2}Q^2 + \mathbb{E}[\frac{(e_i^{all})^2}{2} - Qq(i) + q(i)e_i^{all}|q(i)] \\
&= A - Qq(i) + \mathbb{E}[q(i)e_i^{all}|q(i)]
\end{aligned} \tag{17}$$

where $A = \frac{Q^2}{2} + \mathbb{E}[\frac{(e_i^{all})^2}{2}|q(i)] \leq \frac{Q^2}{2} + \mathbb{E}[\frac{(e_i^{max})^2}{2}|q(i)] = \frac{Q^2}{2} + \frac{(e_i^{max})^2}{2} \triangleq B$.

Accordingly, we have:

$$\triangle(q(i)) \leq B - Qq(i) + \mathbb{E}[q(i)e_i^{all}|q(i)] \tag{18}$$

Based on the above descriptions, we strive to convert the long-term constraint on energy consumption into a per-slot constraint, and minimize a supremum bound on the *drift-plus-penalty* term in each time slot, given as:

$$\begin{aligned}
&\triangle(q(i)) + V\mathbb{E}[t_i^{rl}|q(i)] \\
&\leq B - Qq(i) + \mathbb{E}[q(i)e_i^{all}|q(i)] + V\mathbb{E}[t_i^{rl}|q(i)] \\
&= B - Qq(i) + \mathbb{E}[q(i)e_i^{all} + Vt_i^{rl}|q(i)]
\end{aligned} \tag{19}$$

where $V(>0)$ is used to adjust the preference towards optimization between energy consumption and response latency. A new optimization problem $P2$ is formed, which tries to optimize the right hand side of (19), as below:

$$(P2) \qquad \min_{\forall i, \boldsymbol{I}} \{q(i)e_i^{all} + Vt_i^{rl}\} \tag{20}$$

$$s.t. \qquad (13), (14), (15) \tag{21}$$

Theorem 1: The caching decision $\boldsymbol{I}$ over time slots $\{0, ..., n-1\}$ obtained by solving $P2$ is an approximately optimal solution to $P1$.

Proof Assume that t^{rl*} is the solution to $P1$ and $\boldsymbol{I^*}$ is the corresponding caching decision over the time slots. We have:

$$\begin{aligned}
&\triangle(q(i)) + V\mathbb{E}[t_i^{rl}|q(i)] \\
&\leq B - Qq(i) + \mathbb{E}[q(i)e_i^{all} + Vt_i^{rl}|q(i)] \\
&= B + q(i)\mathbb{E}[(e_i^{all} - Q)|q(i)] + V\mathbb{E}[t_i^{rl}|q(i)] \\
&\overset{\ddagger}{\leq} B + Vt^{rl*}
\end{aligned}$$

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on February 04,2022 at 01:09:13 UTC from IEEE Xplore. Restrictions apply.

The inequality (‡) holds, due to the fact that $e_i^{all} - Q \leq 0$ when $\boldsymbol{I^*}$ is applied. Take the expectation of this inequality and then sum the expectation over the time slots $i \in \{0, ..., n-1\}$,

$$\sum_{i=0}^{n-1} \mathbb{E}[\triangle(q(i)) + V\mathbb{E}[t_i^{rl}|q(i)]]$$
$$= \sum_{i=0}^{n-1} \mathbb{E}[[\mathcal{L}(q(i+1)) - \mathcal{L}(q(i))] + V\mathbb{E}[t_i^{rl}]]$$
$$= \mathbb{E}[\mathcal{L}(q(n)) - \mathcal{L}(q(0))] + \sum_{i=0}^{n-1} V\mathbb{E}[t_i^{rl}]$$
$$= \mathbb{E}[\mathcal{L}(q(n))] + \sum_{i=0}^{n-1} V\mathbb{E}[t_i^{rl}] \leq \sum_{i=0}^{n-1} \mathbb{E}[B + Vt^{rl*}]$$
$$= (B + Vt^{rl*}) * n$$

Owing to $\mathbb{E}[\mathcal{L}(q(n))] \geq 0$, $\sum_{i=0}^{n-1} V\mathbb{E}[t_i^{rl}] \leq (B + Vt^{rl*}) * n$ holds. As a result,

$$\frac{1}{n}\sum_{i=0}^{n-1} t_i^{rl} \leq t^{rl*} + \frac{B}{V}$$

Therefore, the caching decision for optimizing $P2$ can also make the solution of $P1$ infinitely close to the true value (i.e., t^{rl*}) as long as the value of V is properly set. □

B. Algorithm Design

Based on the descriptions, a caching-assisted correlated task offloading algorithm (CCTO) is proposed in Alg. 1 for solving $P1$. During each time slot i, the information about task a_i such as d_i and s_i can be obtained by beacon information exchanging. Then we can calculate the current response latency t_i^{rl} based on the past caching decisions (line 5). Furthermore, the current caching decision can be obtained by minimizing the term $q(i)e_i^{all} + Vt_i^{rl}$. After that, the energy consumption at time slot i can be calculated based on Eq. (10). In the next, we update the energy queue after the constraint condition is checked (line 8-16). Finally, the approximately optimal value V^* can be retrieved.

It shall be noted that the term $q(i)e_i^{all} + Vt_i^{rl}$ totally depends upon e_i^{all}, due to the fact that other variables, such as $q(i)$, V and t_i^{rl}, have been determined with the help of past caching decisions. Obviously, not to cache any computational result can directly minimize e_i^{all}, since the third term $\sum_{j=i-p}^{i} I_j\gamma$ at the right hand of Eq. (10) is zero at any time slot. However, it defeats the object of caching assisted correlated task offloading, i.e., response latency reduction by caching computational results at the past time slots. As shown in Theorem 1, we can actually approximate the true value of $P1$ by adjusting the value of V. Therefore, we can relax the requirement for e_i^{all} minimization. In the meanwhile, we use V to approximate the true value of $P1$, in hope to seek a tradeoff between energy consumption and response latency.

It is noticeable that the response latency for the correlated task offloading is mainly affected by two factors. One is the information of the task itself such as d_i and s_i; and the other is the caching decisions in the past p time slots, which may reduce the workload required for task accomplishment. In view of this, we determine the caching decision at the current time slot (i.e., line 6 in Alg. 1) as follows. Given the task a_i, assume that the edge server has a well memory capability about the past tasks, and the average energy consumption of these tasks plus a_i can be easily calculated, given as $e_i^{avg} = 1/n \sum_{j=0}^{i} e_j^{all}$. For the task a_i, if its energy consumption e_i^{all} satisfies $(e_i^{avg} - Q)/Q < \rho$ and $e_i^{all} < e_i^{max}$, the computational result can be cached at the edge. ρ ($\in (0, 1)$) is used to control the energy queue deviation from Q, which is necessary since we have relaxed the requirements for per-slot energy optimization. An arbitrarily small ρ is helpful for narrowing down the optimality gap between the approximate optimal value and true optimal value.

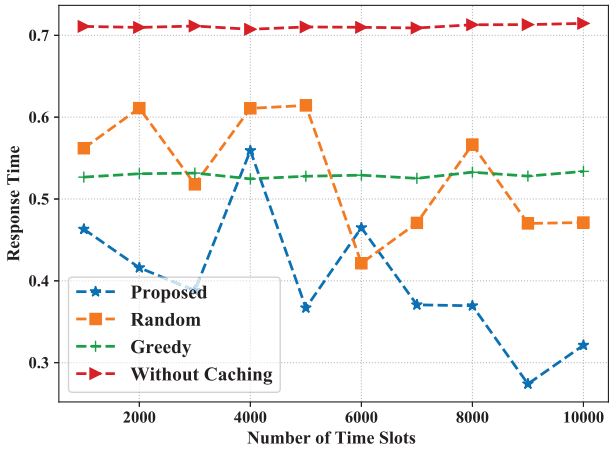

Fig. 1. The average response time with different number of time slots

IV. Numeric Results

We have conducted experiments to evaluate our approach in this section. Two assumptions have been made in the evaluation as follows. First, both d_i and s_i ($0 \leq i \leq n-1$) are evenly distributed over the interval $[d_{min}, d_{max}]$ and $[s_{min}, s_{max}]$, respectively. Second, each task can be completed by the end of its own time slot, so there is no queuing time for any task arriving at the edge server. Note that there are other ways to optimize the per-slot term $q(i)e_i^{all} + Vt_i^{rl}$, such as the random approach and the greedy approach. The former is to randomly cache the computational result while the latter is to cache the result of task which has the minimal energy consumption compared to the last p tasks.

Figure 1 shows the experimental result with regards to the average response time under different time slots. In addition to the aforementioned approaches, i.e., the random approach, denoted by "Random" and the greedy approach, denoted by "Greedy", we also compare our approach with the approach that has not applied any caching strategies, denoted by "Without caching". From this figure, we can easily observe that our approach can averagely achieve the best performance on

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on February 04,2022 at 01:09:13 UTC from IEEE Xplore. Restrictions apply.

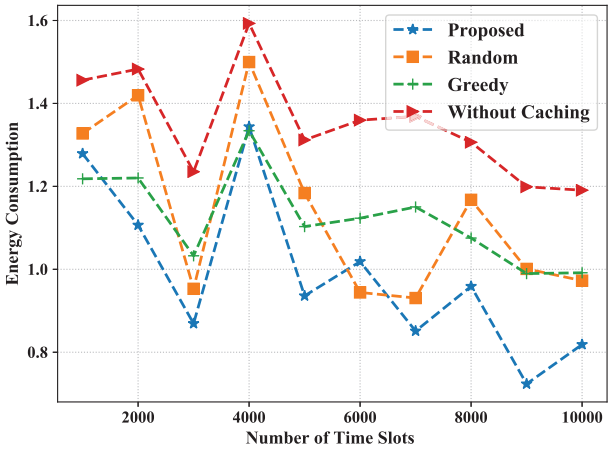

Fig. 2. The average energy consumptions with different number of time slots

optimizing the average response latency. The approach without any caching strategies achieves the worst performance. In the meanwhile, the random approach has a great fluctuation owing to its random caching at each time slot. In addition, the greedy approach does not fluctuate a lot with the increasing number of time slots. Therefore, our approach outstands other three approaches on the response latency optimization.

Figure 2 shows the experimental result with regards to the average energy consumptions under different time slots. Still, our approach achieves the best performance among the four approaches while the approach without caching strategies achieves the worst performance. Similar to the result shown in Fig. 1, the random approach and the greedy approach have shown similar features, respectively. Intuitively, to cache more computational results seems to bring in more energy consumption, which contradicts our simulation result. However, although caching computational results incurs more energy consumption as shown in Eq. 9, the reduced energy consumption on workload can efficiently offset this kind of energy consumption.

V. Conclusion

Caching enabled task offloading for IoT devices in MEC has attracted extensive attention recently. Nevertheless, few of existing works have paid attention to those tasks featured by intrinsic correlations. Indeed, it is difficult to model the intrinsic relationships between two sequential tasks either from coarse granularity or fine granularity. Accordingly, in this paper we develop a general model to depict the correlation relationships between two sequential tasks. The computational result cached at the current time slot can efficiently assist the task performing at future time slots. We evaluate our approach by extensive experiments and the results have shown that the approach indeed outstands other approaches in both response time optimization and energy consumption reduction.

Acknowledgement

This work is partially supported by the National Key R&D Program of China (2020YFB2104301), the Project "Network Communication Intelligent Core Chip Design and Core Software (PCL2021-A08)", and the Project "Beihang Beidou Technological Achievements Transformation and Industrialization Funds (BARI2005)". This work is also partially supported by FCT/MCTES through national funds and when applicable co-funded EU funds under the Project UIDB/50008/2020, and by Brazilian National Council for Research and Development (CNPq) via Grant No. 313036/2020-9. Chunsheng Zhu is the corresponding author.

References

[1] C. Tang, X. Wei, C. Zhu, Y. Wang, and W. Jia, "Mobile vehicles as fog nodes for latency optimization in smart cities," IEEE Transactions on Vehicular Technology, vol. 69, no. 9, pp. 9364–9375, 2020.

[2] C. Zhu, J. J. P. C. Rodrigues, V. C. M. Leung, L. Shu, L. T. Yang, "Trust-Based Communication for the Industrial Internet of Things," IEEE Communications Magazine, vol. 56, no. 2, pp. 16-22, 2018.

[3] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1628–1656, 2017.

[4] Y. Wang, K. Wang, H. Huang, T. Miyazaki, and S. Guo, "Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications", IEEE Transactions on Industrial Informatics, vol.15, no.2, pp.976-986, 2019.

[5] L. Yang, J. Cao, G. Liang and X. Han, "Cost Aware Service Placement and Load Dispatching in Mobile Cloud Systems," IEEE Transactions on Computers, vol. 65, no. 5, pp. 1440–1452, 2016.

[6] J. Xu, L. Chen and P. Zhou, "Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks," IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, Honolulu, HI, 2018, pp. 207–215.

[7] Y. Hao, M. Chen, L. Hu, M. S. Hossain and A. Ghoneim, "Energy Efficient Task Caching and Offloading for Mobile Edge Computing," IEEE Access, vol. 6, pp. 11365–11373, 2018.

[8] H. Xing, J. Cui, Y. Deng and A. Nallanathan, "Energy-Efficient Proactive Caching for Fog Computing with Correlated Task Arrivals," IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Cannes, France, 2019, pp. 1–5.

[9] P. Liu, G. Xu, K. Yang, K. Wang and X. Meng, "Jointly Optimized Energy-Minimal Resource Allocation in Cache-Enhanced Mobile Edge Computing Systems," IEEE Access, vol. 7, pp. 3336-3347, 2019.

[10] X. He, K. Wang, and W. Xu, "QoE-driven content-centric caching with deep reinforcement learning in edge-enabled IoT", IEEE Computational Intelligence Magazine, vol.4, no.4, pp.12-20, 2019.

[11] H. Lu, X. He, M. Du, X. Ruan, Y. Sun, and K. Wang, "Edge QoE: computation offloading with deep reinforcement learning for the Internet of things", IEEE Internet of Things Journal, vol.7, no.10, pp.9255-9265, 2020.

[12] X. -Q. Pham, T. -D. Nguyen, V. Nguyen and E. -N. Huh, "Joint Service Caching and Task Offloading in Multi-Access Edge Computing: A QoE-Based Utility Optimization Approach," IEEE Communications Letters, vol. 25, no. 3, pp. 965–969, March 2021.

[13] Y. Miao, Y. Hao, M. Chen, H. Gharavi and K. Hwang, "Intelligent Task Caching in Edge Cloud via Bandit Learning," IEEE Transactions on Network Science and Engineering, vol. 8, no. 1, pp. 625–637, 2021.

[14] G. Avino, M. Malinverno, F. Malandrino, C. Casetti, and C. F. Chiasserini, "Characterizing Docker Overhead in Mobile Edge Computing Scenarios," In Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems (HotConNet 17), New York, 2017, pp. 30–35.

[15] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," Foundations & Trends in Networking, vol. 1, no. 1, pp. 1–144, 2006.

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on February 04,2022 at 01:09:13 UTC from IEEE Xplore. Restrictions apply.