Task Offloading and Caching for Mobile Edge Computing

Chaogang Tang*, Chunsheng Zhu‡§, Xianglin Wei¶, Huaming Wu†, Qing Li‖, Joel J. P. C. Rodrigues**††
*School of Computer Science and Technology, China University of Mining and Technology, 221116, Xuzhou, China
‡SUSTech Institute of Future Networks, Southern University of Science and Technology, 518055, Shenzhen, China
§PCL Research Center of Networks and Communications, Peng Cheng Laboratory, 518055, Shenzhen, China
¶The 63rd Research Institute, National University of Defense Technology, 410073, Changsha, China
† The Center for Applied Mathematics, Tianjin University, 300072, Tianjin, China
‖ The Hong Kong Polytechnic University, Hong Kong, China
** Federal University of Piauí, Teresina - PI, Brazil
†† Instituto de Telecomunicações, Portugal
cgtang@cumt.edu.cn, chunsheng.tom.zhu@gmail.com, wei_xianglin@163.com,
whming@tju.edu.cn, qing-prof.li@polyu.edu.hk, joeljr@ieee.org

***Abstract*—Mobile applications in the present have created tremendous pressure on the computational capabilities of user equipments. Against this background, mobile edge computing (MEC) has been proposed to tackle this issue, e.g., by shifting the computational workload to the edge server. We in this paper consider a caching enabled task offloading in MEC, for the sake of joint optimization of task offloading and caching. We consider both energy consumption and response latency in the optimization problem and solve the problem by an alternate optimization algorithm. Extensive experiments have been conducted to evaluate the algorithm and the simulation results have shown its advantages such as rapid response latency and powerful convergence capability.**

***Index Terms*—Mobile edge computing, task offloading, task caching, jointly, alternate optimization**

I. Introduction

With the increasing number of smart user equipments (UEs) such as intelligent mobile devices and wearable devices, various mobile applications have created tremendous pressure on the computational capabilities of these UEs. To tackle this issue, mobile edge computing (MEC) has been proposed to ease the pressure of these mobile devices by shifting the computational workload to the edge server deployed at the edge of radio access networks [1] [2]. By doing so, multiple goals has been achieved. For example, on one hand, the high computational resources demand can be mitigated at UEs; on the other hand, for time sensitive applications from UEs, in comparison to task offloading to the remote cloud center, task executed at the mobile edge server can greatly reduce the response time.

In addition, the explosive increment in the number of mobile applications increases the likelihood of multiple offloading for the same tasks. In such cases, frequently duplicated offloading the same tasks to S incurs both large energy consumptions and long response latency. However, such cases can be avoided by caching those most frequently requested tasks [3] [4]. For example, UEs communicate with the edge server by exchanging the beacon information. The edge server can thus retrieve the global information about the requests of all the UEs. They can decide which tasks should be cached according to the popularity and characteristics of these tasks. Once the task a UE wants to offload is cached at the server, the response time can be greatly reduced. Currently, most of works have paid attention to the caching enabled task offloading in MEC such as [5] [6] [7] [8] [9]. For example, authors in [5] proposed to jointly optimize the service offloading and task caching in the long run. To that end, time is slotted and the energy constraint is posed on the optimization objective across over the different time slots. Lyapunov optimization framework is applied to solve the long-term constraint issue. In a caching enabled MEC system, task caching can avoid the offloading of duplicate data. Authors in [10] strive to jointly optimize communication, computation, and caching in MEC systems. They model this problem as a mixed integer non-convex optimization issue. To solve it efficiently, the original problem is decomposed into two subproblems, with each being solved with low-complexity algorithms.

Similar to the aforementioned works, we in this paper also consider caching enabled task offloading in MEC systems. However, the difference lies in that we take into consideration both the energy consumptions at local side and the response latency at the edge server as the optimization objective. In the meanwhile, for part of task offloading in MEC, if the corresponding task is cached at the edge server, the execution result at the local side is also needed to construct an entire computational result together with the part of task execution at the server. In such a case, both the time overheads and energy consumptions also increase, which however has not been considered in most of existing literature.

To be specific, our contributions in this paper can be summarized as follows. First, we propose a generic model to jointly optimize the task offloading and caching in MEC. Second, an alternate optimization technology is applied to this problem by solving the decomposed subproblems respectively. Third, a series of experiments have been conducted to the evaluate

the efficiency and effectiveness of the proposed approach.

The rest of paper is organized as follows. In Section II, a mathematical model is formulated to optimize the task offloading and task caching in MEC such that the overall energy consumption and response latency can be minimized. In Section III, an alternate optimization is applied to solving the two subproblems decomposed from the original problem. Extensive experiments have been reported in Section IV, followed by the conclusion in Section V.

II. System Model

We in this paper consider a system model consisting of one mobile edge server S and n UEs. Assume that each UE has a task for execution. Each UE can choose to offload part of its own task to S with a purpose of mitigating the high demands of computational resources. Each task t_i can be described as a triple $t_i = (d_i, s_i, dl_i)$ where d_i denotes the input data size of t_i which needs to be offloaded over the wireless channels, s_i the number of CPU cycles needed to accomplish t_i and dl_i the deadline for the task accomplishment. As introduced earlier, tasks can be cached at S for future reference, for the reason that most of UEs with same periods and locations usually have similar behaviors to a certain extent. The caching size of S is assumed to be $\mathcal{C}$.

A. Local Execution Model

Owing to the limited computational capabilities of UEs, each UE can reserve part of t_i for local execution. Assume that the ratio of t_i reserved locally is ρ_i, so the local execution time of t_i can be expressed as:

$$t_{loc,i} = \frac{\rho_i s_i}{f_i} \tag{1}$$

where f_i is the processing frequency of UE i. Due to the task offloading, the remaining workload for calculation is $\rho_i s_i$. The corresponding energy consumption for UE i thus can be calculated as [11]:

$$e_{loc,i} = \kappa_i \rho_i s_i f_i^2 \tag{2}$$

where κ_i is the effective capacitance coefficient of the UT's CPU chip [12].

B. Task Offloading Model

When task t_i is partially offloaded to S for execution. The offloading time and corresponding energy consumption can be calculated in what follows. The transmission rate based on Shannon theorem can be given as:

$$r_i = B_i \log(1 + \frac{P_i}{\sigma^2 + I}) \tag{3}$$

where P_i is the transmission power of UE i, B_i is the bandwidth of wireless channel, σ^2 is the noise power and I is the interference caused by other UEs utilizing the same channel as i. Accordingly, the offloading time can be expressed as:

$$t_{off,i} = \frac{(1-\rho_i) d_i}{r_i} \tag{4}$$

Then corresponding energy consumption for task offloading is:

$$e_{off,i} = t_{off,i} \cdot P_i \tag{5}$$

As a result, the total energy consumption at local side can be expressed as

$$e_{sum,i} = e_{loc,i} + e_{off,i} \tag{6}$$

The total energy consumption for t_i at local side should be lower than given threshold, since UEs can be utilized for other purposes. Furthermore, the constraint condition can be expressed as:

$$e_{sum,i} \leq \theta_i \tag{7}$$

where θ_i represents the given threshold for UE i. Generally speaking, the workload for t_i should be executed locally if the total energy consumption exceeds the threshold, otherwise, the workload can be offloaded to S for the sake of energy savings.

C. Task Caching and Execution Model at S

We consider a caching enabled task offloading in mobile edge computing in this paper. Each task can be cached at S for future reference. However, due to the limited caching size of S, it is impractical to cache all the tasks at S. Let variable ϕ_i to denote whether task t_i is cached at S. If t_k is cached, $\phi_i = 1$ and $\phi_i = 0$, otherwise. It is worthwhile mentioning that speaking of task caching we mean that the corresponding execution result are cached. To be specific, if task t_i is cached at S, the computational result of local part at UE i should be offloaded to S where it combines with the offloading part to further construct the complete execution result. The execution time of the offloaded part at S is given as:

$$t_{s,i} = \frac{(1-\rho_i) s_i}{f_s} \tag{8}$$

where f_s is the processing frequency of edger server S.

Task Cached. If task t_i is cached, then the result caching is accomplished until the local result has been retrieved from UE i. Due to the negligible size of execution result compared to the input data size, we omit the time taken to retrieve the local result back from UE i. However, the true accomplishment time for t_i at S should include three parts, i.e., the offloading time for transmitting t_i, the execution time for the offloaded part and the time for local execution result retrieval. Although the third part has been omitted, a partial order rendered by it should be followed. In other words, the true accomplishment time at S should not precede that at local UE. To be specific,

$$t^c_{acc,i} = \max\{t_{off,i} + t_{s,i}, t_{loc,i}\} \tag{9}$$

An implicit assumption above is that the two operations are parallel, i.e., one is the local execution and the other is the offloading action. In addition, there are sufficient computational resources at S, so we assume that each arrived task can be immediately performed by a virtual machine without queuing up.

Task Uncached. If task t_i is not cached, then the result caching is accomplished independent of the local result, which

means that S does not need to retrieve the local execute result from UE i. As a result, the true accomplishment time for t_i at S only includes two parts, i.e., the offloading time for transmitting t_i, the execution time for the offloaded part. Thus, there is no partial order attached on the accomplishment time. The accomplishment time at S can be expressed as:

$$t^n_{acc,i} = t_{off,i} + t_{s,i} \tag{10}$$

To sum up, considering whether the task is cached, the accomplishment time $t_{acc,i}$ can be integratedly expressed as:

$$t_{acc,i} = \phi_i t^c_{acc,i} + (1-\phi_i) t^n_{acc,i} \tag{11}$$

D. Problem Formulation

Before going further, a few assumptions have been made as follows. First, task caching in this paper only involves execution result caching. Actually, many task executions can benefit from the result caching in reality. Second, to evaluate the performance of task caching strategies in mobile edge computing usually needs a long-term process and integrates various factors that influence response latency and energy consumption. Task caching in most of works means to cache the service itself by maintenance of virtual machines (VMs), which however incurs more energy consumption at the edge server. In contrast, we in this paper focus on one type of services of which the computational results are reusable. In the meanwhile, we integrate task caching into task offloading. Third, MEC can shift the computational workload from local side to the edge server, so as to mitigate the energy consumption at UEs. However, in addition to the energy consumption serving as metric to evaluate the performance of caching enabled task offloading in MEC, the response latency for the offloaded tasks is also crucial in MEC, for the reason that the primary reason to apply MEC is to reduce the response latency by pushing the computational resources from the remote cloud center to the edge of networks. Accordingly, we have taken into consideration both the energy consumption at UEs and the response latency for task execution at the edge server.

Thus, the optimization problem can be mathematically as follows.

$$(P) \qquad \min_{\boldsymbol{\phi},\boldsymbol{\rho}} \quad \sum_{i=1}^{n} \{w_1 e_{loc,i} + w_2 t_{acc,i}\} \tag{12}$$

$$\text{s.t.} \quad \mathrm{e_{sum,i}} \le \theta_{\mathrm{i}} \quad \mathrm{i} \in \{1,2,...,\mathrm{n}\} \tag{13}$$

$$\boldsymbol{\rho} = (\rho_1, \rho_2, ..., \rho_n) \tag{14}$$

$$\rho_i \in (0,1) \qquad \forall i \in \{1,2,...,n\} \tag{15}$$

$$\phi_i \in \{0,1\} \qquad \forall i \in \{1,2,...,n\} \tag{16}$$

$$w_1 + w_2 = 1 \quad w_1, w_2 \in (0,1) \tag{17}$$

$$\max\{t_{loc,i}, t^c_{acc,i}, t^n_{acc,i}\} \le dl_i \quad \forall i \in \{1,2,...,n\} \tag{18}$$

where the constraint (13) guarantees that the energy consumption on task execution and offloading should be lower than the specified threshold for each task. w_1 and w_2 are two numeric values to balance the energy saving at UEs and the response time for tasks execution at the edge server. The two values denote the preferences against the energy consumption and response latency when the problem is optimized. Constraint (18) guarantees that each task should be accomplished before the specified deadline.

Challenges. The major difficulty for solving problem P is that it is a mixed discrete-continuous optimization problem with the continuous variable $\boldsymbol{\rho}$ to decide the proportion of one single task that is executed locally and discrete variable $\boldsymbol{\phi}$ to decide which task should be cached at S. To obtain the best solution may take exponential time which is impractical and infeasible in reality.

III. Joint Optimization of Task Offloading and Task Caching

To tackle this issue, we decompose P into two subproblems $P1$ and $P2$. To be specific, $P1$ endeavors to optimize task offloading decision $\boldsymbol{\rho}$ with an assumption that the caching policy $\boldsymbol{\phi}$ is given, and $P2$ endeavors to optimize the task caching profile $\boldsymbol{\phi}$ under the hypothesis of given task offloading decision $\boldsymbol{\rho}$. These two optimization problems are solved alternately in each iteration until convergence condition is achieved. Let $\mathcal{G} = \sum_{i=1}^{n}\{w_1 e_{loc,i} + w_2 t_{acc,i}\}$ and we can obtain:

$$\begin{aligned}
&\mathcal{G}(\boldsymbol{\phi},\boldsymbol{\rho}) = \sum_{i=1}^{n}\{w_1 e_{loc,i} + w_2 t_{acc,i}\} \\
&= \sum_{i=1}^{n} w_1 \kappa_i \rho_i s_i f_i^2 + w_2\{\phi_i t^c_{acc,i} + (1-\phi_i) t^n_{acc,i}\} \\
&= \sum_{i=1}^{n} w_1 \kappa_i \rho_i s_i f_i^2 + w_2\{\phi_i \max\{t_{off,i} + t_{s,i}, t_{loc,i}\} \\
&+ (1-\phi_i)(t_{off,i} + t_{s,i})\} = \sum_{i=1}^{n} w_1 \kappa_i \rho_i s_i f_i^2 \\
&+ w_2\{\phi_i \max\{0, t_{loc,i} - t_{off,i} - t_{s,i}\} + t_{off,i} + t_{s,i}\} \\
&= \sum_{i=1}^{n} w_1 \kappa_i \rho_i s_i f_i^2 + w_2 \phi_i \max\{0, \rho_i(\frac{s_i}{f_i} + \frac{d_i}{r_i} + \frac{s_i}{f_s}) \\
&- \frac{d_i}{r_i} - \frac{s_i}{f_s}\} + w_2\{\frac{d_i}{r_i} + \frac{s_i}{f_s} - \rho_i(\frac{d_i}{r_i} + \frac{s_i}{f_s})\} \\
&= \sum_{i=1}^{n} \rho_i(w_1 \kappa_i s_i f_i^2 - w_2(\frac{d_i}{r_i} + \frac{s_i}{f_s})) \\
&+ w_2 \phi_i \max\{0, \rho_i(\frac{s_i}{f_i} + \frac{d_i}{r_i} + \frac{s_i}{f_s}) - (\frac{d_i}{r_i} + \frac{s_i}{f_s})\} \\
&+ w_2(\frac{d_i}{r_i} + \frac{s_i}{f_s})
\end{aligned} \tag{19}$$

The initial task caching profile is assumed to be $\boldsymbol{\phi}^{(0)}$ and $\boldsymbol{\phi}^{(0)}$ can be established randomly. The problem $P1$ can be formulated as follows:

$$(P1) \qquad \min \mathcal{G}(\boldsymbol{\phi}^0, \boldsymbol{\rho}) \tag{20}$$

$$\text{s.t.} \quad (13)(14)(15)(17)(18) \tag{21}$$

Given task caching decision profile $\boldsymbol{\phi}^{(0)}$, problem $P1$ becomes very easy to solve in terms of $\boldsymbol{\rho}$. For each task t_i, as the offloading profile $\boldsymbol{\rho}$ varies, two cases may take place with regards to $\mathcal{K} = \max\{0, \rho_i(s_i/f_i + d_i/r_i + s_i/f_s) - d_i/r_i - s_i/f_s\}$. To be specific, if $t_{loc,i} > t_{off,i} + t_{s,i}$, $\mathcal{K} = \rho_i(s_i/f_i + d_i/r_i + s_i/f_s) - d_i/r_i - s_i/f_s$; and $\mathcal{K} = 0$, otherwise. Give the task caching profile $\boldsymbol{\phi}^{(0)}$, all the constraints become the linear constraints. Thus, we can further confine ρ_i by solving constraints (13) and (18), respectively. The corresponding result for solving constraint (13) is given as:

$$\rho_i \leq \frac{\theta_i - d_i P_i / r_i}{\kappa_i s_i f_i^2 - P_i d_i / r_i} \tag{22}$$

and the result for solving constraint (18) is given as:

$$\rho_i \leq \frac{dl_i f_i}{s_i} \tag{23}$$

$$\rho_i \geq \frac{d_i/r_i + s_i/f_i - dl_i}{d_i/r_i + s_i/f_i} \tag{24}$$

In addition, let $t_{loc,i} > t_{off,i} + t_{s,i}$, i.e., the execution time of the partitioned task at the local side is larger than the response time of the other part offloaded to the edge server, and we can get:

$$\rho_i \geq \frac{d_i/r_i + s_i/f_s}{s_i/f_i + d_i/r_i + s_i/f_s} \tag{25}$$

Based on the above descriptions, the algorithm called TODM for obtaining the optimal $\boldsymbol{\rho}$ can be developed as shown in Alg. 1

Algorithm 1: Task Offloading Decision Making (TODM)

Input: $\boldsymbol{\phi}^k$, $\boldsymbol{P}$, $\boldsymbol{t}$, $\boldsymbol{\theta}$, σ^2, I, $\boldsymbol{f}$, f_s, w_1, w_2, κ
Output: $\boldsymbol{\rho}$ and $\mathcal{G}$

$g = 0$;
for each t_i in task set **do**
  $\rho_i^1 = \frac{\theta_i - d_i P_i/r_i}{\kappa_i s_i f_i^2 - P_i d_i / r_i}$;
  $\rho_i^2 = \frac{dl_i f_i}{s_i}$;
  $\rho_i^3 = \frac{d_i/r_i + s_i/f_i - dl_i}{d_i/r_i + s_i/f_i}$;
  $\rho_i^4 = \frac{d_i/r_i + s_i/f_s}{s_i/f_i + d_i/r_i + s_i/f_s}$;
  $\rho_i^5 = 0$ and $\rho_i^6 = 1$;
  Select ρ_i^* from $\boldsymbol{\rho} = \{\rho_i^1, ..., \rho_i^6\}$ by solving $\rho_i^* = \arg\min\{w_1 e_{loc,i} + w_2 t_{acc,i}\}$;
  $g = g + \min\{w_1 e_{loc,i} + w_2 t_{acc,i}\}$;
end
Construct $\boldsymbol{\rho} = (\rho_1^*, \rho_2^*, ..., \rho_n^*)$;
return $\boldsymbol{\rho}$, g;

TODM can make an offloading decision with regards to $\boldsymbol{\rho}$ in real time since its time complexity is of $O(n)$. It is worthwhile mentioning that the global objective value g is constructed based on each task.

TODM runs with $\boldsymbol{\phi}^0$ as the parameter, and thus we can obtain the task offloading decision $\boldsymbol{\rho}^0$. Now it comes to the second subproblem $P2$ which strives to optimize the caching decision profile $\boldsymbol{\phi}$ given $\boldsymbol{\rho}^0$. Specifically, the subproblem $P2$ can be formulated as:

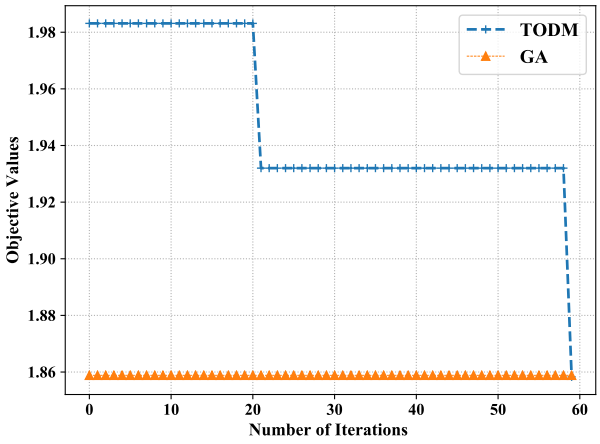

Fig. 1. The convergence comparison between TODM and GA

$$(P2) \qquad \min \mathcal{G}(\boldsymbol{\phi}, \boldsymbol{\rho}^0) \tag{26}$$

$$\text{s.t.} \quad (13)(17)(18) \tag{27}$$

The subproblem $P2$ is actually a knapsack problem with the decision variable $\boldsymbol{\phi}$. Exhaustive searching is impractical in reality due to the NP hardness of this problem. Therefore, in this paper, we propose to solve $P2$ by GA. As one of swarm intelligence algorithms, GA is featured by simple deployment, fast convergence and so on.

The two subproblems $P1$ and $P2$ can be alternately solved in each iteration until convergence condition is achieved. To be specific, the procedure of solving problem P can be shown in Alg. 2.

Algorithm 2: Joint Optimization of Task Offloading and Task Caching (TOTC)

Construct $\boldsymbol{\phi}^0$;
$k = 0$;
while $\Delta g > \epsilon$ **do**
  `// Solve subproblem P1`
  $\boldsymbol{\rho}^k, g_1 = TODM(\boldsymbol{\phi}^k)$;
  Obtain $\boldsymbol{\rho}^{k+1}, g_2$ by GA;
  `// Solve subproblem P2`
  $\Delta g = |g_2 - g_1|$;
  $k = k + 1$;
end
return $\boldsymbol{\rho}$, $\boldsymbol{\phi}$, g;

IV. Experimental Evaluation

A. Experimental Settings

In this section, a series of experiments have been carried out to evaluate the joint optimization of task offloading and task caching in MEC proposed in this paper. Before going

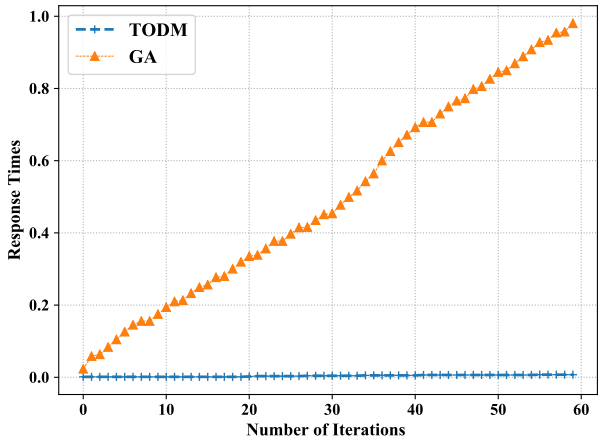

Fig. 2. The response time comparison with the increasing number of iterations

further, the experimental setting ups are introduced in what follows. Each UE decides to offload a task S for execution and each task is randomly generated with d_i varying from 10 to 30, s_i from 10 to 60, dl_i from 0 to 1. The number of UEs is 20. For the GA involved parameters, the crossover and mutation probability are set to 0.2 and 0.02, respectively. The population size and the number of iterations are 100 and 300, respectively. The chromosome length is the number of UEs. In additions, w_1 and w_2 are set to 0.7 and 0.3 respectively.

B. Simulation Results and Analysis

At the beginning, we investigate the convergence capacity of TOTC. TOTC alternately applies TODM and GA for minimizing the problem P. The experimental result is shown in Fig. 1 where the x coordinate denotes the number of iterations and the y coordinate denotes the objective values with regards to problem P. During each iteration, TODM and GA record the currently best $\boldsymbol{\rho}$, $\boldsymbol{\phi}$ and the best objective value. As the number of iteration increases, TODM and GA tend to converge to the optimal value as shown in this figure. For example, when the number of iterations is 60, both TODM and GA are convergent to one point. It is worth noticing that the task offloading decision $\boldsymbol{\rho}$ has little influence on GA when it strives to obtain the task caching profile since the best objective value obtained by GA seems unchanged as the number of iterations increases. On the other hand, from the figure, TODM has shown relatively powerful convergence capability.

Second, the response time of TODM and GA is recorded in Fig. 2 where the x coordinate represents the number of iterations and the y coordinate represents the response time. From the figure, we can obviously observe that TODM can make a realtime response no matter how the number of iterations increases. However, the response time of GA rapidly increases with the increasing number of iterations. To sum up, GA can obtain a better objective value at the beginning compared to TODM at the expense of relatively long response time.

V. Conclusion

Extensive works have focused on task offloading and task caching in MEC systems, in hope to reduce the energy consumption at the local UEs on one hand and reduce the response latency on the other hand. We in this paper have proposed a optimization model to jointly optimize task offloading and task caching in MEC with regards to energy consumption and response latency. To solve this problem including both the continuous variable (i.e., task offloading decision) and discrete variable (i.e., task caching decision), an alternate optimization algorithm has been put forward to solve the problem.

Acknowledgment

This work is partially supported by the National Key R&D Program of China (2020YFB2104301), the project "PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications (LZC0019)", 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/EEA/50008/2020, and by Brazilian National Council for Research and Development (CNPq) via Grant No. 309335/2017-5. Chunsheng Zhu is the corresponding author.

References

[1] H. Wu, Y. Sun, and K. Wolter, "Energy-efficient decision making for mobile cloud offloading," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 570–584, 2020.

[2] Y. Deng, Z. Chen, X. Yao, S. Hassan, and J. Wu, "Task scheduling for smart city applications based on multi-server mobile edge computing," *IEEE Access*, vol. 7, pp. 14 410–14 421, 2019.

[3] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," *Computer Networks*, vol. 57, no. 16, pp. 3128–3141, 2013.

[4] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 3, pp. 1473–1499, 2015.

[5] 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*, 2018, pp. 207–215.

[6] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoniem, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, pp. 11 365–11 373, 2018.

[7] 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.

[8] 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*, ser. HotConNet '17. Association for Computing Machinery, 2017, pp. 30–35.

[9] Z. Qin, S. Leng, J. Zhou, and S. Mao, "Collaborative edge computing and caching in vehicular networks," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, 2020, pp. 1–6.

[10] 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.

[11] 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.

[12] C. Tang, C. Zhu, X. Wei, H. Wu, Q. Li, and J. J. P. C. Rodrigues, "Intelligent resource allocation for utility optimization in rsu-empowered vehicular network," *IEEE Access*, vol. 8, pp. 94 453–94 462, 2020.