Received: 17 February 2021 | Revised: 6 May 2021 | Accepted: 17 June 2021

DOI: 10.1002/cpe.6494

RESEARCH ARTICLE

WILEY

# Task migration computation offloading with low delay for mobile edge computing in vehicular networks

**Bingxue Qiao**[1] | **Chubo Liu**[1] | **Jing Liu**[2] | **Yikun Hu**[1] | **Kenli Li**[1] | **Keqin Li**[3]

[1]College of Information Science and Engineering, Hunan University, Hunan, China

[2]Department of Computer Science, Wuhan University of Science and Technology, Wuhan, China

[3]Department of Computer Science, State University of New York, New York, New Paltz, USA

**Correspondence**
Chubo Liu, College of Information Science and Engineering, Hunan University, and National Supercomputing Center in Changsha, Hunan 410082, China.
Email: liuchubo@hnu.edu.cn

## Abstract

Nowadays, a new paradigm named mobile edge computing (MEC) is capable of supplying some cloud-like functions at the edges of wireless networks, which enables vehicles to offload the computation intensive tasks on MEC servers with low latency. However, new challenges posed by the complex network environment and the mobility of vehicles are usually not covered by traditional offloading schemes. To solve such problems, we propose a heuristic task migration computation offloading (TMCO) scheme. Compared with traditional ones, TMCO can dynamically choose suitable places to offload the tasks for moving vehicles within deadline. For this purpose, the mobility of vehicle and strict delay deadline are considered comprehensively. We use hash table to store the number of tasks on the corresponding server and use random function to simulate the probability of task offloading. In terms of latency, experimental results suggest that the performance of TMCO is on average 10% higher than that of traditional full offloading schemes.

**KEYWORDS**

delay, Internet of Vehicles, mobile edge computing, task offloading

## 1 | INTRODUCTION

### 1.1 | Motivation

As Internet of Vehicles (IoV) evolves, various vehicular applications, for example, autonomous driving,[1] driver-safety improvement,[2] and traffic control,[3] have emerged to enhance the safety and efficiency of vehicles,[4] and therefore the quality of experience (QoE) of drivers and passengers. However, the demanding resources are often not satisfied with mobile vehicles.[5,6] Such circumstances call for new solutions to handle the mounting demands for computing and QoE. Mobile edge computing (MEC) has emerged as one of the most promising solutions that can meet the ever-growing QoE requirements and computation demands.[7-11]

MEC is aimed at providing some cloud-like functions for nearby users with low latency by deploying limited computing and storage resources close to them. With MEC, resource-hungry mobile devices (MDs) can offload delay-sensitive jobs to MEC servers for execution,[12] which satisfies the requirements of expanding computing capabilities of MDs, as well as reducing excessive delays compared with remote cloud services.

However, in the IoV environment, the differences between vehicular networks and the ordinary static scenes result in some new problems. For example, due to the mobility of vehicle, it may drive out of MEC server's coverage before the task is processed completely on it, which makes it more difficult to specify an appropriate offloading scheme. In this article, let us assume that the trajectory is known of the vehicles in a fixed area. As shown in Figure 1, area *a* has multiple base stations, each one covering only a limited range. Suppose that each vehicle is associated with a task to be processed and during the task offloading phase, it can only be transferred to the server that covers it. In this fixed area, if the traffic is dense, the multiple computational tasks are very likely processed in the same MEC server. As a consequence, the probability of task congestion and

  

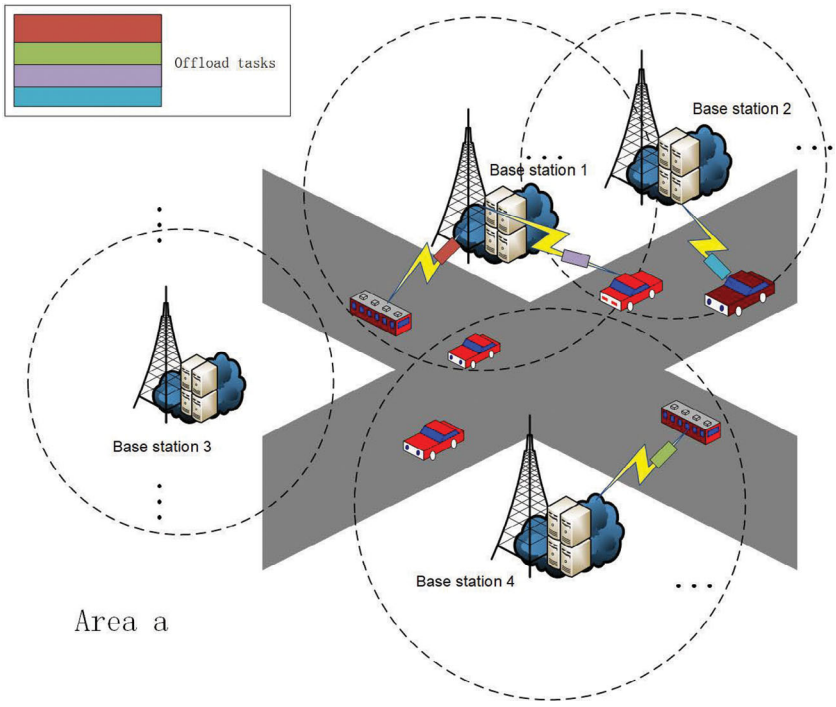


**FIGURE 1** Architecture of mobile edge computing system in Internet of Vehicles

queuing delay dramatically rises, leading to the reduction of users' QoE. For sake of solving this problem, we put forward a heuristic task migration computation offloading (TMCO) scheme. In TMCO, tasks can be computed either locally or remotely. When a task is processed remotely, a relatively suitable server is selected to provide services. With this algorithm, we can effectively reduce the average delays of tasks.

## 1.2 | Related work

Computation offloading is critical in MEC, which determines computation efficiency and achievable performance.[13] Offloading computing tasks to MEC servers helps to prolong the battery lifetime of MDs and improve the QoE of users.[14-18]

Recently, various offloading schemes that pay attention to the latency optimization have been proposed in the literature.[19-23] Yang et al.[19] studied the partition of computing on cloud resources and the scheduling offloading computing. In Reference 20, Sun and Ansari proposed the layout strategy of cloudlet network. In Reference 21, Sun and Ansari reduced the average delay of tasks that arrived at the MEC server based on the Poisson process. Mao et al. regarded the delay as a part of their measurements in Reference 22, and reduced the delay by optimizing the measurement. For the purpose of optimizing execution delay, Jia et al. proposed a model which is used to capture the response times of offloaded tasks in Reference 23.

However, the above works don't take the strict delay deadline into account. In view of this, there are several works focus on the issue.[24,25] In Reference 24, Chen et al. used none-cooperative game to build the model so as to achieve this objective. In the case of delay constraint, Sardellitti et al. tried to reduce the sum of energy consumption of MDs.[25] Nevertheless, the aforementioned works do not take into account the mobility of the device.

When involving mobility in MEC, it becomes more difficult to make appropriate computing offloading scheme for low delay. There are mainly three methods of mobility management.[26] The first one is power control, the second is virtual machine (VM) migration, and the last one is path selection. In this article, we use VM migration scheme to deal with the mobility of the vehicles. There have been a number of studies on the mobility of MDs.[27-30] Wang et al.[27] provided a Q-learning based mobility management policy with partial server-side information. In Reference 28, Xu et al. studied the partial information scenario, and they introduced service migration cost. Ding et al.[29] developed different service migration strategies depending on mobility types. In Reference 30, Sun et al. used Lyapunov to optimize the computation delay. However, the works do not consider the appropriate offloading scheme whether the task is processed in the local or the MEC server when multiple tasks need to be processed. So they also have their own limits.

In view of this, we propose the TMCO algorithm for IoV in the MEC with mobility consideration. In TMCO, we consider not only the mobility of vehicles but also the strict delay deadline. By making the offloading decision for each task, the average delay of the tasks is optimized. The existing works we have summarized are shown in Table 1.

**TABLE 1** Comparisons between proposed algorithm and other offloading schemes

| Schemes | Mobility | Deadline | Objective(s) |
|---|---|---|---|
| Sun and Ansari[20] | Yes | No | Average delay |
| Sun and Ansari[21] | No | No | Average delay |
| Mao et al.[22] | No | No | Dealy and energy |
| Chen et al.[24] | No | Yes | Delay and energy |
| Sardellitti et al.[25] | No | Yes | Energy |
| Sun et al.[30] | Yes | No | Average delay |
| Task migration computation offloading | Yes | Yes | Average delay |

### 1.3 | Contributions

Although MEC is expected to improve users' mobile experience, it is tricky to establish a suitable task offloading model with the MEC architecture. When the mobility is considered in computation offloading phase, the problem becomes more difficult. In this article, our aim is to reduce computing task offloading delays during the process of vehicles' movement. Specifically, for each task, we make an offloading decision by comparing the completion delay of the task locally and remotely. In particular, when the task is processed on the MEC server, it will migrate to different servers and the appropriate server is selected to provide services.

The main contributions of this article are listed as follows.

- We consider the mobility of vehicles in computation offloading stage and dynamically make offloading decisions during the task offloading process to reduce the delay.
- We strictly guarantee the deadlines of offloaded tasks compared with existing works. By taking the deadline of task into account, users' QoE can be improved effectively.
- We characterize the performance of TMCO, prove it can achieve an optimal offloading, and demonstrate the impact of the task deadline and the number of MEC servers on offloading strategy by extensive experiments.

This article is organized as follows. Section 2 introduces the system model. Section 3 presents the dynamic partial offloading algorithm. In Section 4, extensive simulation experiments are done to evaluate the algorithm we proposed. Section 5 concludes the article and mentions our future work.

## 2 | SYSTEM MODEL AND PROBLEM FORMULATION

In this part, our system model and corresponding optimization problem are formally formulated. The definitions of the notations used in this article are summarized in Table 2.

### 2.1 | Architecture model

In Figure 1, there is a known area $a$ with multiple deployed base stations and each base station deploys a MEC server responsible for the provision of services for nearby vehicles. Denote $M$ as the number of MEC servers and $\mathcal{M} = \{1, 2, \dots, M\}$ as the corresponding server set.

Suppose there are multiple vehicles moving in area $a$, and each vehicle has a task to be processed. Denote $N$ as the number of vehicles and $\mathcal{N} = \{1, 2, \dots, N\}$ as the corresponding task set. In the rest of this article, $\mathcal{N}$ is used to indicate the sets of vehicles and tasks interchangeably. In this work, we know the trajectory of the vehicles in area $a$. Therefore, if a task is processed remotely, it will be first transmitted to a base station and then migrated to different servers that can provide services within the deadlines and the completion delays are compared in each server. Finally, the server with the least completion is selected as the work server.

### 2.2 | System model

Let the parameter $\mu$ indicate whether the task is executed on the MEC server. Therefore, $\mu$ can only take two values, 0 (not offload to) and 1 (offload to). For each computation task $n$, we use a three tuple $\langle \lambda_n, \gamma_n, d_n \rangle$ to describe it, where $\lambda_n$ (in bit) is the amount of input data for $n$, $\gamma_n$ (CPU cycles/bit) indicates how many CPU cycles needed to calculate one bit of input data, and $d_n$ is the deadline for task $n$.

**TABLE 2** Mathematical symbols

| Notation | Definition |
|---|---|
| $M$ | Number of MEC servers |
| $N$ | Number of tasks |
| $\mu$ | Offloading decision |
| $\lambda_n$ | Input data size |
| $\gamma_n$ | CPU cycles in bits |
| $d_n$ | Deadline of tasks |
| $f_n^l$ | Local calculation frequency |
| $f_m^c$ | MEC capacity |
| $D_n^l$ | Local execution delay |
| $\sigma_m$ | Noise power |
| $H_{n,m}$ | Channel gain |
| $p_n$ | Power bound |
| $B$ | Bandwidth |
| $R_n^m$ | Data rate |
| $D_t(n,m)$ | Transmission delay |
| $D_n^e$ | Remote execution delay |
| $m_n$ | The server that provides services finally |
| $\tau_m$ | Number of tasks offloaded to server $m$ |
| $n^m$ | Task execution sequence |
| $h_n$ | Handover cost |
| $H$ | One-time handover cost |
| $\mathcal{A}(n)$ | Serving MEC servers for task $n$ |
| $S_n$ | Start time |
| $C_n$ | Completion time |
| $D_n$ | Total delay |
| $T_{\text{num}}^m$ | Number of tasks in server $m$ |
| $D_q(n,m)$ | Queue delay |
| $\chi$ | Random waiting probability |
| $E_n^m$ | Total execution time |
| $T_m$ | Task migration rate |

Abbreviation: MEC, mobile edge computing.

### 2.2.1 | Local computing

Denote $f_n^l$ as the clock frequency of the vehicle executing task $n$. Here, different tasks are allowed to have different clock frequency. We obtain the execution delay in local as

$$D_n^l = \frac{\lambda_n \gamma_n}{f_n^l}. \tag{1}$$

### 2.2.2 | Remote computing

Before executing tasks at edge servers, the task should be transmitted to the base stations. Suppose that task $n$ is offloaded to the base station $m \in \mathcal{M}$, and the power consumption for transmitting the input data of task $n$ is denoted as $p_n$. Then the achievable rate of task $n$ to base station $m$ can be expressed as

$$R_{n,m} = B \log_2 \left(1 + \frac{p_n H_{n,m}}{\sigma_m}\right), \tag{2}$$

where $H_{n,m}$ is the channel gain between the vehicle and base station. $\sigma_m$ is the white noise power level, $B$ is the channel bandwidth. Then we can get that the transmission delay is

$$D_t(n,m) = \frac{\lambda_n}{R_{n,m}}. \tag{3}$$

Denote $f_m^c$ as the clock frequency of the MEC server $m$, and $m_n \in \mathcal{M}$ is the server that provide services finally. Then the execution time is given by

$$D_n^e = \frac{\lambda_n \gamma_n}{f_{m_n}^c}. \tag{4}$$

### 2.2.3 | Handover and migration cost

Due to the mobility of the vehicle, task $n$ may not be processed in the MEC server that is originally offloaded, that is, we should consider the problem of the migration of servers. When the task is not processed in the initial MEC server, there will be additional delay costs because of task handover process and computation migration. Denote $H$ as the one-time handover cost for task $n$. As mentioned above, task $n$ is finally processed on the MEC server $m_n$. We assume that the task passes through $m_{\text{num}}$ servers from start to finish in total. Denote $\mathcal{A}(n) = (m_n^1, m_n^2, \ldots, m_n^{m_{\text{num}}})$, $\mathcal{A}(n) \subseteq \mathcal{M}$ is the set of serving MEC servers for task $n$. Then we can get the overall handover cost for task $n$, that is

$$h_n = H \sum_{p=2}^{m_{\text{num}}} \Pi\{m_n^p \neq m_n^{p-1}\}, \tag{5}$$

where

$$\Pi\{x\} = \begin{cases} 1, & x \text{ is true;} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Let $\tau_m$ be the number of tasks which are offloaded to MEC server $m$, and $n^m = (n_1^m, n_2^m, \ldots, n_{\tau_m}^m)(n_1^m, n_2^m, \ldots, n_{\tau_m}^m \in \mathcal{N})$ be the task execution sequence in server $m$, which satisfies $D_t(n_1^m, m) \leqslant D_t(n_2^m, m) \leqslant \ldots \leqslant D_t(n_{\tau_m}^m, m)$, that is, each MEC server maintains a first-in-first-out queue to cache waiting tasks and adopts the first-come-first-served queuing discipline for arrival tasks. Then, the start time of task $n$ can be represented as

$$S_{n_l^m} = \begin{cases} D_t(n_l^m, m) + h_{n_l^m}, & \text{if } l = 1; \\ \max\{D_t(n_l^m, m) + h_{n_l^m}, C_{n_{l-1}^m}\}, & \text{if } l > 1, \end{cases} \tag{7}$$

where $C_{n_{l-1}^m}$ is the completion time of the previous task, and the completion time of task $n$ can be expressed as

$$C_n = S_n + D_n^e. \tag{8}$$

It is assumed that the sizes of result datas are small, so the feedback delays are ignored.

## 2.3 | Problem formulation

Though the above analysis, the delay of task $n$ can be calculated as

$$D_n = \min\{D_n^l, C_n\}. \tag{9}$$

In area $a$, there are multiple vehicles and each vehicle is associated with a task. Some tasks of them are executed in MEC servers but others in the local. As mentioned above, we use the parameter $\mu$ to decide the offloading decision for each task. Here, denote the vector $\Phi_n = (\mu_1, \mu_2, \ldots, \mu_n)$ as the offloading strategy of all tasks, in which the value of each variable is the integer zero or the integer one. In this work, we need to find a suitable

$\Phi_n$ to optimize the average delay of all tasks when resources and vehicle mobility are limited. The problem is formally described in mathematical as follows:

$$P: \text{minimize} \frac{1}{N}\sum_{n=1}^{N} D_n, \tag{10}$$

$$\text{s.t.} D_n \leqslant d_n, \tag{11}$$

$$\mathcal{A}(n) \subseteq \mathcal{M}, \mu_n \in \{0, 1\}. \tag{12}$$

Constrain (11) indicates that task $n$ must be completed before its deadline, and constrain (12) indicates that the worker server is in the area $a$ and the value range of parameter $\mu$.

## 3 | ALGORITHM IMPLEMENTATION

Based on the formulation analysis of the problem $P$, this article proposes TMCO algorithm to the problem $P$ above. The pith idea of the TMCO algorithm is to diminish the average delay of task completion through making appropriate offloading decisions for each task. In this section, our algorithm is analyzed.

### 3.1 | Problem analysis

For task $n$, the process of computing in the local is expressed as *process*($a$) and in the server as *process*($b$). As shown in Figure 2, the dotted line indicates that task $n$ is processed locally by on-board equipment, and the solid line represents the processing process of the task in the servers. In *process*($a$), task $n$ is finally processed in the local, which is relatively simple to compute the completion delay because there is no task migration. However, In *process*($b$), the process of task execution becomes more complex. Suppose that base stations 1–4 all can serve task $n$. Initially, task $n$ is offloaded to base station, after a series of migrations, it is finally processed in the base station 4, which is the most optimal server for task $n$ compared with other servers.

In this article, since the moving track of the vehicle is known, we can determine which MEC servers the task can migrate to.

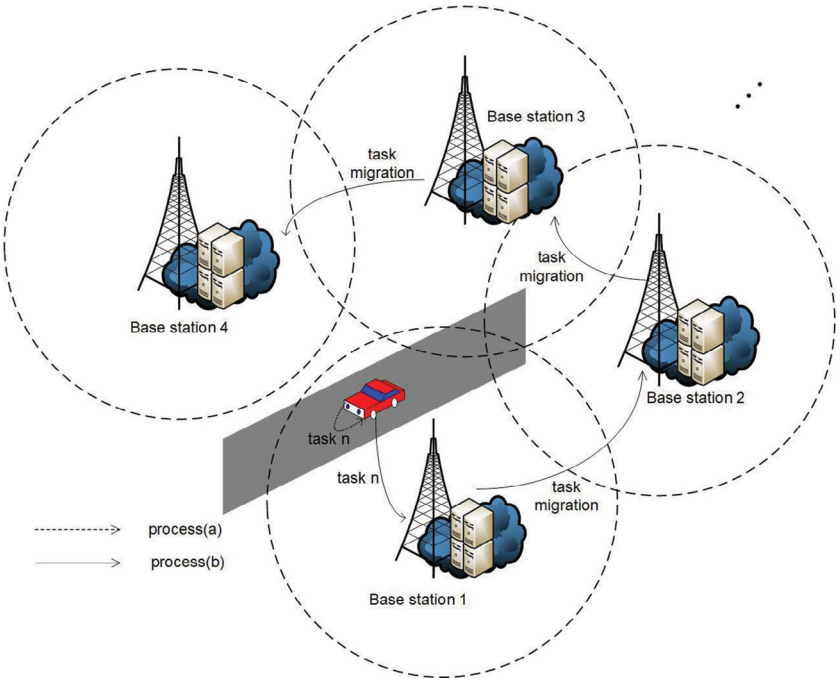


**FIGURE 2** Task processing locally or remotely

Our goal is to find the suitable offloading scheme for each task. In the initial state, let each value in $\Phi_n$ be zero, that is, $\mu_1 = \mu_2 = \ldots = \mu_n = 0$. We use hash table to describe the servers, denote the two-tuples of parameters $\langle m, T^m_{\text{num}} \rangle$ as the server's state, where $m$ is the unique identifier of the MEC server $m$ and $T^m_{\text{num}}$ is the number of tasks in $m$. In the initial state, $T^m_{\text{num}} = 0$, which means that when the first task is offloaded to the MEC server, it does not need to queue.

The two-tuples $\langle m, T^m_{\text{num}} \rangle$ changes dynamically with the offloading of the tasks. Specifically, only when a task is determined to be executed on a server, the number of tasks on the server will increase, that is, if a task is eventually processed in the server $m$, the corresponding $T^m_{\text{num}}$ increases by one, the current $T^m_{\text{num}}$ can be expressed as $T^m_{\text{num}} = T^m_{\text{num}} + 1$.

Due to the mobility of vehicles, when a task is offloaded to the server during task migration, we cannot determine how many tasks are waiting to be executed before it. Therefore, we use a random parameter $\chi$ to indicate the probability of waiting for a task to be offloaded to a server. Then we can get that the queue delay of task $n$ in MEC server $m$ is

$$D_q(n,m) = \begin{cases} 0, & T^m_{\text{num}} = 0; \\ \chi D^e_{n^m_1}, & T^m_{\text{num}} = 1; \\ \chi D^e_{n^m_1} + \sum_{l=2}^{T^m_{\text{num}}} D^e_{n^m_l}, & T^m_{\text{num}} > 1, \end{cases} \tag{13}$$

where $\chi \in [0, 1]$. Due to the uncertainty of task execution, in our article, we use random function to generate the value of $\chi$.

*Hash table*. It is a data structure that can be accessed directly according to the key value. Specifically, it can access records by mapping key value to a location in the table, thus speeding up the search. In the hash table, the value of key cannot be repeated. In our work, we use the data structure to store the unique identifier of MEC server and the number of tasks executed in the server.

*Random function*. It can generate random numbers between zero and one. In this work, we use it to randomly generate the implementation progress of task in the MEC server. When the number zero is generated, it means that the task has not yet started to execute and when the number one is generated, it means that the task has been completed. Denote the parameter $\zeta$ as the number randomly generated by random function, then the $\chi$ can be expressed as $\chi = 1 - \zeta$.

After the above calculation, we are able to get the execution delay of task in different MEC servers under the current situation. Then we need to choose the most optimal server as the one to provide services. Assume that $M_a$ is a matrix of order one times $m$, in which each location stores the execution time $E^m_n$ of the task in the corresponding server. As mentioned in section 2.2.2, the server $m_n$ is the one that provide services finally. Then we have

$$E^{m_n}_n = 1 \Big/ \left\| \frac{1}{h_n + D_q(n,m) + \frac{\lambda_n \gamma_n}{f^c_m}} \right\|_1, \tag{14}$$

where $\|\mathcal{E}\|_1$ indicates 1-norm of the matrix $M_a$ and $\mathcal{E}$ represents the reciprocal of the task's execution time in the server.

When the task is processed locally, from Equation (1), we can get the completion delay for task $n$ in the on-board equipment local is $D^l_n = \frac{\lambda_n \gamma_n}{f^l_n}$. When processing in the server $m_n$, the completion delay of the task is given by

$$\begin{aligned} C_n &= S_n + D^e_n \\ &= D_t(n, m_n) + E^{m_n}_n \\ &= \frac{\lambda_n}{R_{n,m_n}} + 1 \Big/ \left\| \frac{1}{h_n + D_q(n,m) + \frac{\lambda_n \gamma_n}{f^c_m}} \right\|_1. \end{aligned} \tag{15}$$

Through the above calculation results, we can get the offloading decision as follows:

$$\mu_n = \begin{cases} 1, & D^l_n \geqslant C_n; \\ 0, & D^l_n < C_n. \end{cases} \tag{16}$$

For every task, repeat the above steps. Then we can get the offloading scheme of all tasks in area $a$, the average completion delay of them can be given by $D_{\text{average}} = \frac{1}{N} \sum_{n=1}^{N} D_n$.

In this work, we use task migration rate to indicate the proportion of tasks offloaded to the MEC server. Denote the parameter count as the number of the tasks offloaded to the MEC server. In the initial state, let count = 0. In the calculation process, if task is processed in the MEC server finally, the variable count plus one, that is, count $\leftarrow$ count + 1. Then, we can get the task migration rate is $T_m = \frac{1}{N}$count.

### 3.2 | Task migration computation offloading

The TMCO algorithm is based on the idea of task migration. When there are many tasks waiting to be executed on a server, the queuing delay will be greatly increased, leading to the reduction of users' QoE. Therefore, we propose TMCO algorithm. In the TMCO, the task can be processed in the local or server, when it is computed in the MEC server, it will choose the one that takes the least time as the service server. Though this algorithm, we can find out a vector of optimal offloading strategies, which can reduce the average delay of all tasks effectively.

In TMCO, we first use an array to represent the offload status of the tasks, which stores Boolean values. Value 0 represents that the task is processed in the local, and value 1 represents that the task is processed in the MEC server. In the initial state, we make all values in this array be zero. In the following process, update the array according to calculation results. For the server, we use a hash table to represent how many computing tasks are on a particular server. The key-value pair represents the server identity name and the number of tasks calculated on it, respectively. Whenever a task is executed on it, the number of tasks on the corresponding server is need to add one. When a task is migrated to a server for execution, take a parameter $\chi$ between zero and one to indicate the completion progress of the task in the server. Then, the queue delay of the current task is calculated according to the parameter $\chi$.

Through the above series of calculations, we can get the minimum delay of task execution in server, and then we compare it with the local execution delay. If the task takes less time locally, the value of the task in the offloading policy array is unchanged and remains zero. Conversely, the value in the array is updated to one. The detailed algorithm steps are shown in Algorithm 1.

**Algorithm 1.** TMCO algorithm

**Require:** $\lambda_n, \gamma_n, d_n, f_n^l, B, p_n, H_{n,m}, \sigma_m, f_m^c, H_m$;
**Ensure:** $\Phi_n$, average completion delay of tasks, and task migration rate;

**for** each task $n \in N$ **do**
  $\mu_n \leftarrow 0$;
**end for**
**for** each MEC server $m \in M$ **do**
  $T_{\text{num}}^m \leftarrow 0$;
**end for**
**for** $n \leqslant N$ **do**
  **for** $m \leqslant M$ **do**
    /*Completion delay in the local*/
    $D_n^l \leftarrow \frac{\lambda_n \gamma_n}{f_n^l}$;
    /*Completion delay in the MEC server*/
    $C_n \leftarrow D_t(n, m) + E_n^{m_n}$;
    $D_n \leftarrow \min\{D_n^l, C_n\}$;
  **end for**
  /*Choose the server $m_n$ that takes the least time*/
  $T_{\text{num}}^{m_n} \leftarrow T_{\text{num}}^{m_n} + 1$;
**end for**
**for** $n \leqslant N$ **do**
  /*Total delay*/
  $D_{\text{all}} \leftarrow 0$;
  $D_{\text{all}} \leftarrow D_{\text{all}} + D_n$;
  $count \leftarrow 0$;
  **if** $\mu_n = 1$ **then**
    $count \leftarrow count + 1$;
  **end if**
  /*Compute the average delay of all tasks*/
  $D_{\text{average}} \leftarrow \frac{1}{N} D_{\text{all}}$;
  /*Compute the task migration rate*/
  $T_m \leftarrow \frac{1}{N} count$;
**end for**

### 3.3 | Complexity

In this section, time complexity of TMCO algorithm is presented. Through Algorithm 1, we need to initialize the value of $\mu$, which requires time complexity $\Theta(N)$ (Steps 1–3). $M$ is the number of MEC servers, we need to set the initial number of tasks on each MEC server to zero, which takes time $\Theta(M)$ (Steps 4–6). In the third loop (Steps 7–17), for each task, we need to dynamically compare the current state of each server and the local computing time to choose where the task is executed. Hence, this loop takes time $\Theta(NM)$. Finally, in Steps 18–30, according to the execution status of each task, we calculate the average delay of all tasks and task migration rate, which tasks time $\Theta(N)$. Base on above analysis, we can get the time complexity of TMCO algorithm is $\Theta(N + M + NM + N)$.

## 4 | EXPERIMENT ANALYSIS

In this section, simulation results are carried out to demonstrate the effectiveness of the TMCO algorithm and the performance of it is evaluated.

In our work, we consider the way to optimize the average delay in the process of moving tasks in a known area. The moving tasks can be computed in the local or server. So, the following two schemes are chosen as comparison algorithms. One is to consider that all tasks are executed locally, and in this case, vehicles have no impact on task processing. This scheme is named local offloading process (LOP) algorithm. The other is to consider processing all tasks in the MEC server, under this circumstances, the migration of tasks is still thought about but tasks are not allowed to be processed locally, and this scheme is named full offloading (FO) algorithm.

### 4.1 | Parameter setup

In the simulation experiments, we observe the results by adjusting $N$ and $M$. As Table 3 shows, the number of tasks ranges from 100 to 1000. The number of MEC servers ranges from 1 to 100. The input data size of a computing task is between 100 and 500 kb, its execution request is selected from [500, 1500] cycles/bit, and its deadline is varied from 5 to 20 s. The local calculation frequency is fixed as 0.1 GHz. Denote a value in the set $\{0.5, 0.6, \ldots, 1.0\}$ GHz[31] as the CPU frequency ($f_m^c$) for each MEC server. The noise power ($\sigma_m$) is denoted as $10^{-9}$ W,[20] and the channel gain ($H_{n,m}$) is $10^{-7}$. The power bound ($p_n$) is 1 W, and the channel bandwidth $B$ is set to 1 MHz.[32]

### 4.2 | Comparison results

The relationship between the average delay of task completion and the number of tasks and servers is shown in Figure 3. As seen in this figure, with the growth of the tasks, the average completion delay of tasks do not change with the use of LOP algorithm, but will increase by using the other two schemes. This is because in this work, we considered that when the same task is being calculated, the local computing frequency is fixed. And while

**TABLE 3** System parameters

| System parameters | Varied range |
|---|---|
| Number of tasks ($N$) | [100, 1000] |
| Number of MEC servers ($M$) | [1, 100] |
| Input data size ($\lambda_n$) | [100, 500] kb |
| Execution request ($\gamma_n$) | [500, 1500] cycles/bit |
| Deadline of tasks ($d_n$) | [5, 20] s |
| Local calculation frequency | 0.1 GHz |
| MEC capacity ($f_m^c$) | {0.5, 0.6, … , 1.0} GHz |
| Noise power ($\sigma_m$) | $10^{-9}$ W |
| Channel gain ($H_{n,m}$) | $10^{-7}$ |
| Power bound($p_n$) | 1 W |
| Bandwidth ($B$) | 1 MHz |

Abbreviation: MEC, mobile edge computing.

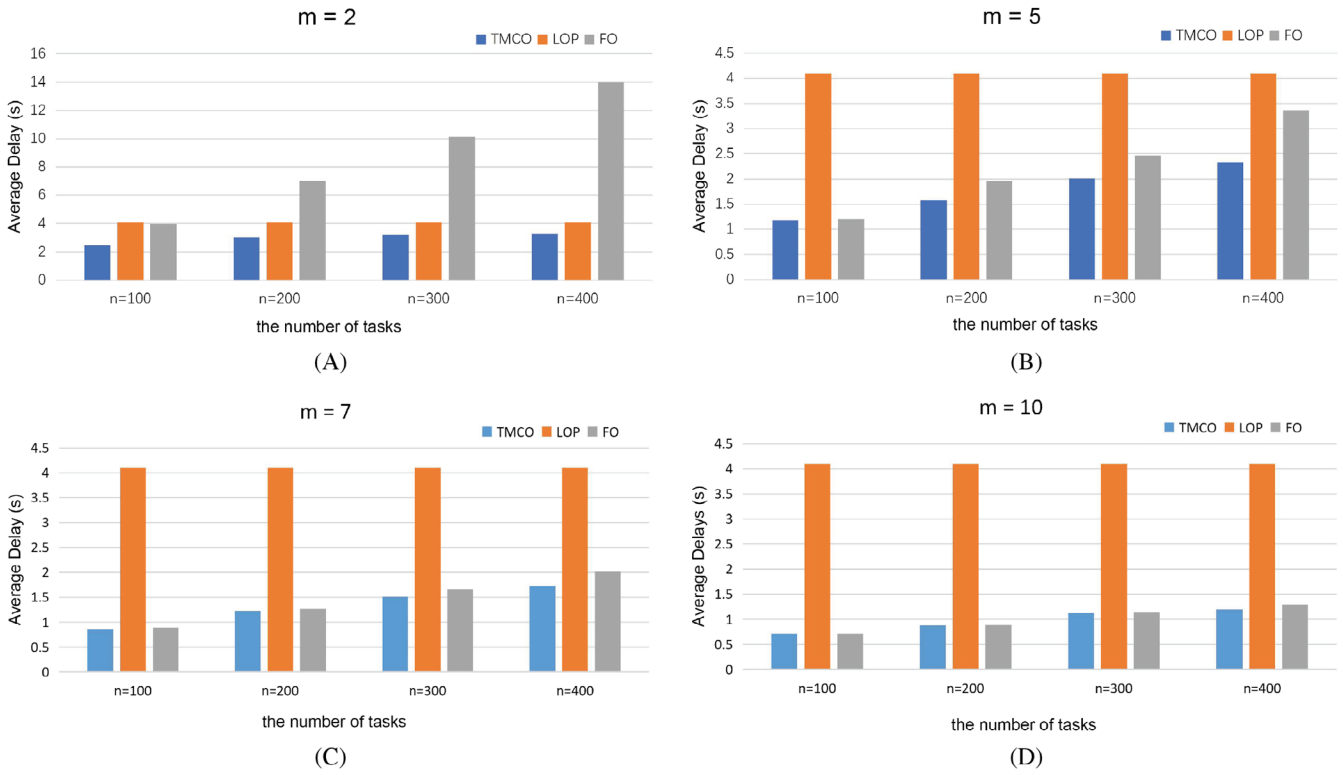


**FIGURE 3** Average delay of all tasks

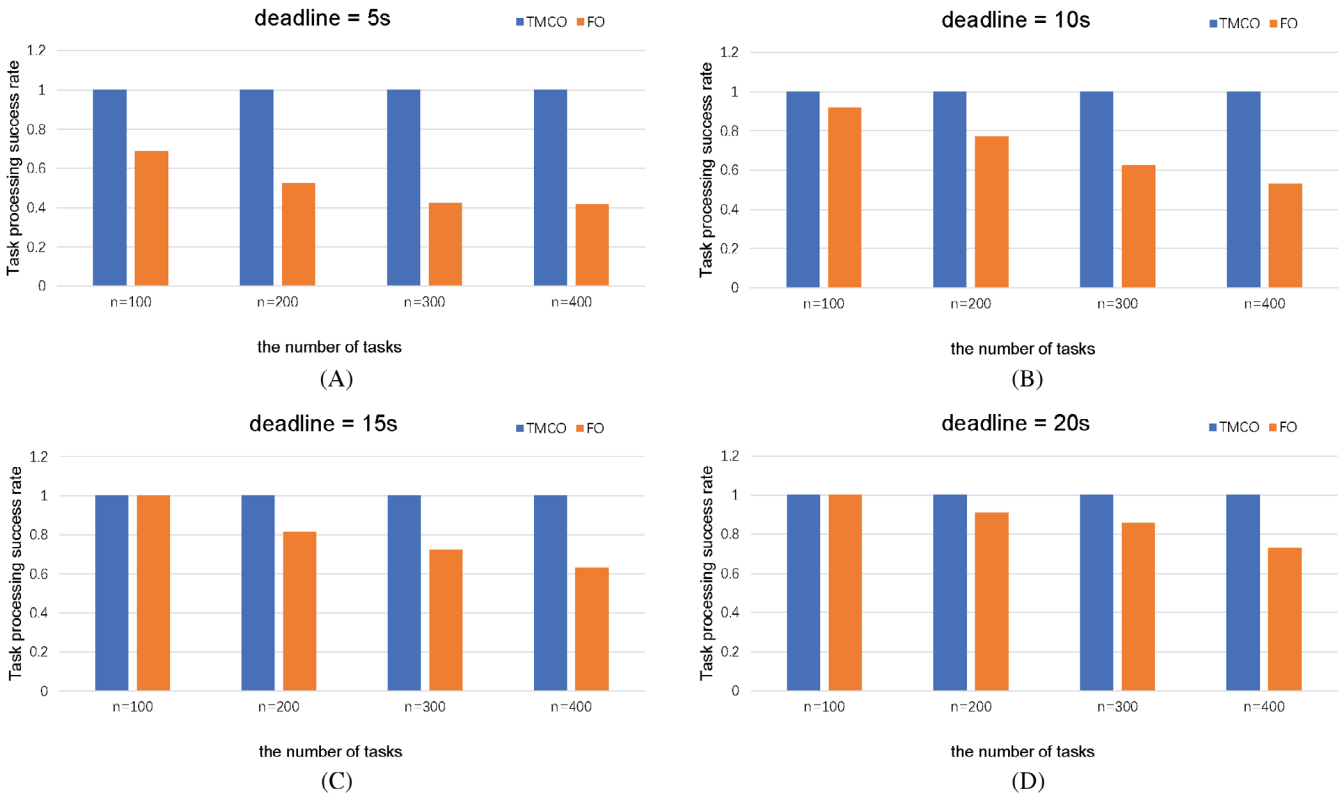


**FIGURE 4** Success rate of task processing

the task is offloaded to the server for remote execution, because the number of MEC servers is fixed, the more the number of tasks is, the longer the queue delay of subsequent tasks, the greater the average completion delay of the task.

By comparing the four subfigures (A)–(D) in Figure 3, it can be seen that when there are only two servers, with the increase of the tasks, the average of the FO scheme is much higher than that of the other two schemes, but with the increase of servers, FO scheme takes less average completion delay, LOP becomes the most time-consuming scheme among the three schemes. The reason is that when the number of servers is too small, if all tasks are executed on the server, as the number of tasks grows, the queuing delay of tasks to be executed will greatly increase, thereby increasing the average completion delay of all tasks. As the number of servers increases, more servers can choose to be offloaded, and the queuing delay is greatly reduced. At this time, the average completion delay of all tasks in scheme TMCO and FO is much lower than that in scheme LOP.

It can be seen that the success rate of task processing between the proposed algorithm and FO algorithm under different deadlines is compared in Figure 4. In this work, we suppose that all tasks can be successfully processed in the local process. Hence, in the figures, we do not draw the result of LOP algorithm. By comparing the four graphs (A)–(D) in Figure 4, we know that TMCO can always handle all tasks successfully. As for FO algorithm, with the increase of tasks, the success rate of task processing reduces, which is due to the increased queuing delay of tasks on each server. When the number of tasks is the same, the larger the deadline, the greater the success rate of task processing. In addition, FO algorithm can even achieve the same success rate as TMCO under the case of a small amount of input data and a long deadline.

Figure 5 shows the task migration rate of three schemes mentioned above. It is manifest from the figure that the task migration rate of LOP and FO are 0% and 100%, respectively. For TMCO algorithm, as the number of tasks grows, the task mobility decreases. This is because while the number of servers is fixed, the more the number of tasks, the longer the queuing delay, thus increasing the average completion delay of all tasks. In the TMCO algorithm, we always choose the best place to process tasks, so the task migration rate is reduced.

Figure 6 shows the relationship between task migration rate and the number of tasks and servers. It is observed that along with the number of MEC servers increases, the task migration rate reduces. The reason for this has mentioned above, so we do not repeat it here. In addition, as the number of MEC servers increases, the decline of task migration rate is more and more gentle. The reason is that computing capabilities of the server is far greater than that of the local vehicle. When there are enough MEC servers to provide task-execution services, tasks will be preferentially executed on the server.

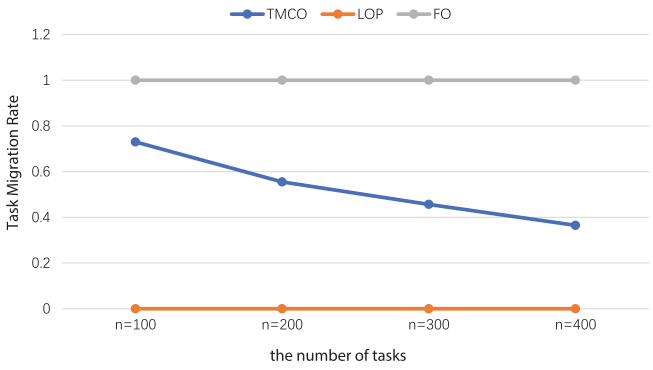


**FIGURE 5** Task migration rate of three schemes

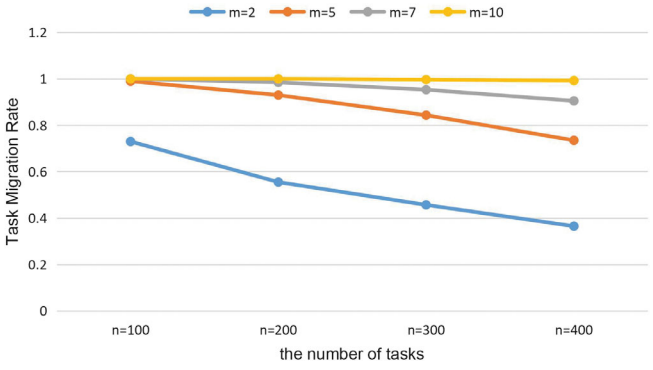


**FIGURE 6** Relationship between task migration rate and the number of tasks and servers

## 5 | CONCLUSIONS

MEC technology has become increasingly significant in our daily life and work with the growing demand of reducing computing delay. Our research focuses on the issue of computing offloading scheme with low latency under the case of mobility management and strict deadline in the MEC-based IoV. By dynamically adjusting the offloading strategy of computing tasks, we can choose to process them in the local or edge servers. Finally, an appropriate set of offloading strategies can be selected for all tasks. Compared with traditional task offloading schemes, experiment results show that the algorithm proposed in this article has a good performance improvement in delay consumption. In the future, we will study in depth the scenarios that the number of servers and network conditions are unknown.

### ACKNOWLEDGMENTS

The authors would like to express their gratitude to the anonymous reviewers whose constructive comments have greatly helped to improve this article. This work was partially supported by the National Key Research and Development Program of China (Nos. 2018YFB0204302, 2018YFB1701403), Programs of National Natural Science Foundation of China (Grant Nos. 62072165, U19A2058). This work was also sponsored by Zhejiang Lab, China (No. 2020KE0AB01) and the Key Project of Scientific Research Plan of Hubei Provincial Department of Education (Grant No. D20201102).

### DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

### ORCID

*Bingxue Qiao* https://orcid.org/0000-0003-2002-0546
*Kenli Li* https://orcid.org/0000-0002-2635-7716

### REFERENCES

1. Du H, Leng S, Wu F, Chen X, Mao S. A new vehicular fog computing architecture for cooperative sensing of autonomous driving. *IEEE Access*. 2020;8:10997-11006.
2. Choi Y, Han SI, Kong SH, et al. Driver status monitoring systems for smart vehicles using physiological sensors: a safety enhancement system from automobile manufacturers. *IEEE Signal Process Mag*. 2016;33(6):22-34.
3. C. Feng, W. Jing, Deadline-constrained data aggregation scheduling in urban vehicular networks, in: 19th IEEE International Conference on e-Health Networking, Applications and Services, Healthcom 2017, Dalian, China, October 12-15, 2017, IEEE, 2017, pp. 1–5.
4. Hou X, Ren Z, Wang J, et al. Reliable computation offloading for edge-computing-enabled software-defined IoV. *IEEE Internet Things J*. 2020;7(8):7097-7111.
5. Zhu C, Tao J, Pastor G, et al. Folo: latency and quality optimized task allocation in vehicular fog computing. *IEEE Internet Things J*. 2019;6(3):4150-4161.
6. Kang J, Yu R, Huang X, et al. Blockchain for secure and efficient data sharing in vehicular edge computing and networks. *IEEE Internet Things J*. 2019;6(3):4660-4670.
7. Yan KT, Yu MP, Tran NH, et al. Energy-efficient resource management in UAV-assisted mobile edge computing. *IEEE Commun Lett*. 2021;25(1):249-253.
8. Ali SSD, Zhao HP, Kim H. Mobile edge computing: a promising paradigm for future communication systems, in: TENCON 2018 - 2018 IEEE Region 10 Conference, Jeju, South Korea, October 28 - 31,2018, IEEE, 2018, pp. 1183 - 1187.
9. Tang F, Liu C, Li K, et al. Task migration optimization for guaranteeing delay deadline with mobility consideration in mobile edge computing. *J Syst Archit*. 2020;112(8):101849.
10. Saleem U, Liu Y, Jangsher S, Li Y, Jiang T. Mobility-aware joint task scheduling and resource allocation for cooperative mobile edge computing. *IEEE Trans Wirel Commun*. 2021;20(1):360-374.
11. Li S, Wang Q, Wang Y, et al. Joint congestion control and resource allocation for delay-aware tasks in mobile edge computing. *Wirel Commun Mob Comput*. 2021;2021(1):1-16.
12. Chiang M, Tao Z. Fog and IoT: an overview of research opportunities. *IEEE Internet Things J*. 2017;3(6):854-864.
13. Chang Z , Zhou X , Wang Z , et al. Edge-assisted adaptive video streaming with deep learning in mobile edge networks, in: 2019 IEEE Wireless Communications and Networking Conference, WCNC 2019, Marrakesh, Morocco, April 15 - 18, 2019, IEEE, 2019, pp. 1-6.
14. Mehrabi M, You D, Latzko V, et al. Device-enhanced MEC: multi-access edge computing (MEC) aided by end device computation and caching: a survey. *IEEE Access*. 2019;7(99):166079-166108.
15. Lee S, Lee S, MK. Shin. Low cost MEC server placement and association in 5G networks, in: 2019 International Conference on Information and Communication Technology Convergence, ICTC 2019, Jeju Island, Korea (South), October 16-18, 2019, IEEE, 2019, pp. 879–882.
16. Gonzalez-Sosa E, Frontelo-Benito I, Kachach R, Perez P, Ruiz JJ, Villegas A. Audience meter: a use case of deploying machine learning algorithms over 5G networks with MEC, in: 2020 IEEE International Conference on Consumer Elec- tronics (ICCE), Las Vegas, NV, USA, January 4-6, 2020, IEEE, 2020, pp. 1–2.
17. A. Hairuman, A. Zahra, G. P. Kusuma and D. F. Murad, εMEC Deployment with Distributed Cloud in 4G Network for 5G Success,ε 2019 6th International Conference on Information Technology, Computer and Electrical Engineering (ICITACEE), Semarang, Indonesia, 2019, pp. 1-6.
18. Liu B, Liu C, Peng M. Resource allocation for energy-efficient MEC in NOMA-enabled massive IoT networks. *IEEE J Select Areas Commun*. 2020;1.
19. Yang L, Cao J, Cheng H, Ji Y. Multi-user computation partitioning for latency sensitive mobile cloud applications. *IEEE Trans*. 2015;64(8):2253-2266.
20. Sun X, Ansari N. PRIMAL: profit maximization avatar placement for mobile edge computing, in: 2016 IEEE International Conference on Communications, ICC 2016, Kuala Lumpur, Malaysia, May 22-27, 2016, IEEE, 2016, pp. 1–6.

21. Xiang S, Ansari N. Latency aware workload offloading in the cloudlet network. *IEEE Commun Lett*. 2017;21(7):1481–1484.
22. Mao Y, Zhang J, Tetaief KB. Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems, in: 2017 IEEE Wireless Com- munications and Networking Conference, WCNC 2017, San Francisco, CA, USA, March 19-22, 2017, IEEE, 2017, pp. 1–6.
23. Jia M, Liang W, Xu Z Huang M; Cloudlet load balancing in wireless metropolitan area networks, in: 35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016, IEEE, 2016, pp. 1–9.
24. Chen X, Jiao L, Li W, Fu X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans Network*. 2016;24(5):2795-2808.
25. Sardellitti S, Scutari G, Barbarossa S. Joint optimization of radio and computational resources for multicell mobile cloud computing, in: 2014 IEEE 15th International Workshop on Signal Processing Advances in Wireless Communica- tions (SPAWC), Toronto, ON, Canada, June 22-25, 2014, IEEE, 2014, pp. 354–358.
26. Mach P, Becvar Z. Mobile edge computing: a survey on architecture and computation offloading. *IEEE Commun Surv Tutor*. 2017;19(3):1628–1656.
27. Wang J, Liu K, Li M, Pan J. Learning based mobility management under uncertainties for mobile edge computing, in: IEEE Global Communications Confer- ence, GLOBECOM 2018, Abu Dhabi, United Arab Emirates, December 9-13, 2018, IEEE, 2018, pp. 1–6.
28. Xu J, Sun Y, Chen L, Zhou S. E2M2: energy efficient mobility management in dense small cells with mobile edge computing, in: IEEE International Conference on Communications, ICC 2017, Paris, France, May 21-25, 2017, IEEE, 2017, pp. 1–6.
29. Ding Y, Liu C, Li K, Tang Z, Li K. Task offloading and service migration strategies for user equipments with mobility consideration in mobile edge computing, in: 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking, ISPA/BDCloud/SocialCom/SustainCom 2019, Xia- men, China, December 16-18, 2019, IEEE, 2019, pp. 176–183.
30. Sun Y, Zhou S, Xu J. EMM: energy-aware mobility management for mobile edge computing in ultra dense networks. *IEEE J Select Areas Commun*. 2017;35(11):2637–2646.
31. Liu C, Li K, Liang J, et al. COOPER-MATCH: job offloading with a cooperative game for guaranteeing strict deadlines in MEC. *IEEE Trans Mob Comput*. 2019;1–1.
32. Liu C, Li K, Liang J, et al. COOPER-SCHED: a cooperative scheduling framework for mobile edge computing with expected deadline guarantee. *IEEE Trans Parallel Distrib Syst*. 2019;1–1.

**How to cite this article:** Qiao B, Liu C, Liu J, Hu Y, Li K, Li K. Task migration computation offloading with low delay for mobile edge computing in vehicular networks. *Concurrency Computat Pract Exper*. 2022;34(1):e6494. https://doi.org/10.1002/cpe.6494