Delay-Aware Task Scheduling for Multi-Access Edge Computing on the Internet of Vehicles

You-Chiun Wang and Kuan-Yu Chen
Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
Email: ycwang@cse.nsysu.edu.tw; yoyo126358@gmail.com

***Abstract*—The demand for computing and networking in cars has grown with the advancement of the *Internet of vehicles (IoV)*. Using *multi-access edge computing (MEC)* can deal with the issue of high latency in cloud computing. However, fast movement of cars and limited resources of MEC servers brings challenges. As a car moves into a cell (i.e., handoff), its MEC server may have no enough resources to serve the car's task. Therefore, the paper proposes a *delay-aware task scheduling (DTS)* scheme. When an MEC server has resources in short supply, some tasks are selected to be offloaded to nearby MEC servers. If a car is about to leave a cell, its task is offloaded to the MEC server in the car's target cell. Otherwise, we choose MEC servers for offloading based on their resources, bandwidth, and serving tasks. Simulation results reveal that the DTS scheme can improve the service ratio while lowering the response latency.**

***Keywords*—delay, handoff, IoV, MEC, scheduling.**

I. Introduction

The *Internet of vehicles (IoV)* interconnects cars, sensors, devices, and the Internet through mobile networks (e.g., 5G). Common applications include road safety, traffic management, driving assistance, and infotainment services, which have different delay demands [1], [2]. In the past, cloud computing was viewed as a key computing technique for IoV task execution, but it incurs network congestion and high latency problems [3]. *Multi-access edge computing (MEC)* is a promising solution, which extends capabilities of cloud computing to the edge of a mobile network. It delivers computing and storage resources in the proximity of users [4]. MEC servers are often co-located with *base stations (BSs)* to support computation-intensive or data-intensive applications with low latency. We call them *BS and MEC server pairs (BMPs)*, as shown in Fig. 1.

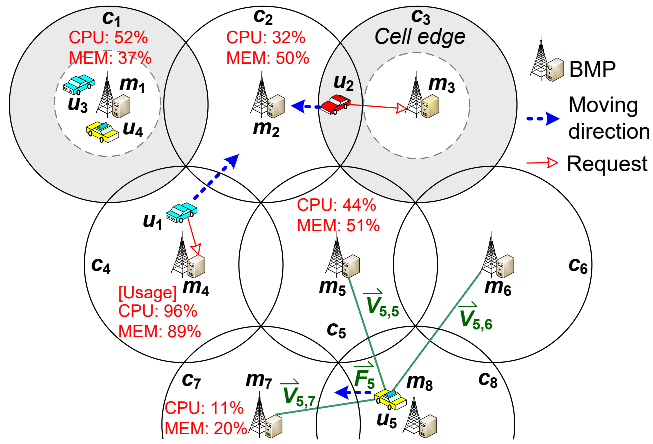


Fig. 1. Task offloading in an MEC-enabled mobile network.

As cars move at high speeds, the handoff frequency mounts. Some MEC servers may not have sufficient resources to serve new tasks when cars enter their cells. This could cause long response latency or even tasks failed, making a great impact on certain applications (e.g., road safety). One feasible solution is to transfer some tasks of heavy-loaded MEC servers to others (referred to as *offloading*). How to select MEC servers to offload is a challenge. Greedily picking the one that has the most resources is not necessarily the best choice. Consider Fig. 1 as an example. A car u_1 is in cell c_4, so it selects BMP m_4 for service. Although m_4's BS has enough *resource blocks (RBs)* to serve u_1, m_4's MEC server has insufficient CPU resources. Suppose that we want to offload u_1's task to a neighboring MEC server. The greedy method picks the MEC server in BMP m_7 to offload. However, since u_1 moves towards cell c_2, the communication path is m_7 (MEC server) $\rightarrow m_2$ (BS) $\rightarrow u_1$. Doing so significantly raises the response latency, which may make packets dropped due to expiration. If fact, a good choice is to select the MEC server in m_2 for offloading.

In this paper, we thus design a *delay-aware task scheduling (DTS)* scheme. When an MEC server does not possess enough resources to handle a new task, DTS picks out some tasks to be offloaded, taking account of cars' locations and tasks' delay budgets. If a car is moving to another cell soon, its task will be transferred to the MEC server of the target cell. Otherwise, DTS selects nearby MEC servers for offloading according to multiple factors, such as residual resources, bandwidth, and the number of serving tasks. The *geographic information system (GIS)* is employed to find target cells for cars. When GIS is not available, we predict target cells using vector calculation. With simulations, we display that the DTS scheme can increase the service ratio and decrease the response latency efficiently.

II. Related Work

Some MEC issues related to IoV have been discussed. The work [5] implements service migration among MEC servers by using Docker containers. In [6], software-defined networking is applied to keep service continuity for cars when they move between cells. The study [7] proposes a handover method over MEC servers. These studies have different objectives with ours.

A few studies copy task data to other MEC servers before cars move to their cells (referred to as *service replication*). The study [8] compares service migration and replication using an analytical model. The work [9] designs a replication method based on the number of unsuccessful tasks and the ratio of read and write tasks. In [10], an integer linear problem for service replication is defined with an aim to reduce the response time of tasks. Dai *et al.* [11] propose a heterogeneous replication approach with distributed convex relaxation. However, service replication may induce a high storage cost for MEC servers.

To find MEC servers to offload tasks, a convolution neural network is used in [12] to analyze car trajectories, while the work [13] gathers users' Wi-Fi traces to discover association patterns. Both [14] and [15] handle the offloading problem via deep reinforcement learning. However, they may transfer tasks to busy MEC servers. Given one grid-based road network, the work [16] delivers tasks to MEC servers to reduce costs and ensure response latency. The study [17] picks MEC servers for offloading using TOPSIS (technique for order of preference by similarity to ideal solution) [18]. In [19], the policy gradient mechanism is adopted to find MEC servers for offloading.

Compared to previous studies, our work considers not only delay budgets of tasks but also multiple factors of MEC servers for offloading (e.g., resources, bandwidth, and serving tasks). This can raise the service ratio and reduce the response latency.

III. System Model

Let us consider one mobile network with a set $\hat{C}$ of cells, as Fig. 1 shows. Each cell $c_i \in \hat{C}$ has a BMP m_i whose BS offers RBs and MEC server supplies CPU and *memory (MEM)* resources. Let $r_i^{\texttt{RB}}$, $r_i^{\texttt{CPU}}$, and $r_i^{\texttt{MEM}}$ be the residual numbers of RB, CPU, and MEM resources of m_i. There is also a set $\hat{U}$ of cars. Each car $u_j \in \hat{U}$ requires a rate λ_j for communications and may issue a task $\psi_j = (d_j^{\texttt{CPU}}, d_j^{\texttt{MEM}}, s_j, \tau_j)$, where $d_j^{\texttt{CPU}}$ and $d_j^{\texttt{MEM}}$ are the requested numbers of CPU and MEM resources, s_j is ψ_j's data size, and τ_j is the delay budget.

Suppose that a car u_j is now in cell c_i and will move into cell c_l after completing processing ψ_j. If BMP m_i offloads ψ_j to BMP m_k, the overall flow for ψ_j is $u_j \Rightarrow m_i \Rightarrow m_k \Rightarrow m_l \Rightarrow u_j$. Let $\tilde{B}(x, y)$ be the bandwidth between two items x and y (e.g., car or BMP). The flow contains five steps, where the amount of time taken by each step is estimated as follows:

1. Request sending ($u_j \Rightarrow m_i$)**:** $t_1 = \rho_j^{\texttt{REQ}}/\tilde{B}(u_j, m_i)$, where $\rho_j^{\texttt{REQ}}$ is the length of request packet for ψ_j.

2. Task offloading ($m_i \Rightarrow m_k$)**:** $t_2 = s_j/\tilde{B}(m_i, m_k)$.

3. Task handling (at m_k**):** $t_3 = s_j/(d_j^{\texttt{CPU}} \times \varepsilon) + \beta$. Here, ε is the processing capability of a unit of CPU resource (in bits/s), and β is the delay due to context switch and resource competition (when m_k has multiple tasks). According to [20], β is about a few microseconds and rises as m_k has more tasks.

4. Result passing ($m_k \Rightarrow m_l$)**:** $t_4 = s'_j/\tilde{B}(m_k, m_l)$, where s'_j is the size of processing result for ψ_j's data.

Algorithm 1: The DTS Scheme

1 **foreach** $u_j \in \hat{U}$ **do**
2   estimate the number of RBs $d_j^{\texttt{RB}}$ for u_j to meet λ_j;
3   **if** $r_i^{\texttt{RB}} < d_j^{\texttt{RB}}$ **then**
4   continue;
5   $r_i^{\texttt{RB}} \leftarrow r_i^{\texttt{RB}} - d_j^{\texttt{RB}}$;
6   **if** $r_i^{\texttt{CPU}} \geq d_j^{\texttt{CPU}}$ *and* $r_i^{\texttt{MEM}} \geq d_j^{\texttt{MEM}}$ **then**
7     $r_i^{\texttt{CPU}} \leftarrow r_i^{\texttt{CPU}} - d_j^{\texttt{CPU}}$, $r_i^{\texttt{MEM}} \leftarrow r_i^{\texttt{MEM}} - d_j^{\texttt{MEM}}$;
8   continue;
9   $r_i'^{\texttt{CPU}} \leftarrow r_i^{\texttt{CPU}} - d_j^{\texttt{CPU}}$, $r_i'^{\texttt{MEM}} \leftarrow r_i^{\texttt{MEM}} - d_j^{\texttt{MEM}}$;
10   $\Psi_j \leftarrow \text{OffloadTask}(\hat{T}_i \cup \{\psi_j\}, m_i)$;
11   sort tasks in Ψ_j decreasingly by their sizes;
12   **foreach** $\psi_x \in \Psi_j$ **do**
13     $m_k \leftarrow \text{TargetBMP}(\psi_x, m_i)$;
14   **if** $m_k \neq$ *null* **then**
15   mark ψ_x;
16       $r_i'^{\texttt{CPU}} \leftarrow r_i'^{\texttt{CPU}} + d_x^{\texttt{CPU}}$, $r_i'^{\texttt{MEM}} \leftarrow r_i'^{\texttt{MEM}} + d_x^{\texttt{MEM}}$;
17   **if** $r_i'^{\texttt{CPU}} \geq 0$ *and* $r_i'^{\texttt{MEM}} \geq 0$ **then**
18   break;
19   **if** $r_i'^{\texttt{CPU}} < 0$ *or* $r_i'^{\texttt{MEM}} < 0$ **then**
20   drop ψ_j and continue;
21   **foreach** *marked task* ψ_x *in* Ψ_j **do**
22   offload ψ_x to its selected BMP m_k;
23     $r_k^{\texttt{CPU}} \leftarrow r_k^{\texttt{CPU}} - d_x^{\texttt{CPU}}$, $r_k^{\texttt{MEM}} \leftarrow r_k^{\texttt{MEM}} - d_x^{\texttt{MEM}}$;
24   $r_i^{\texttt{CPU}} \leftarrow r_i'^{\texttt{CPU}}$, $r_i^{\texttt{MEM}} \leftarrow r_i'^{\texttt{MEM}}$;

5. Reply sending ($m_l \Rightarrow u_j$)**:** $t_5 = \rho_j^{\texttt{REP}}/\tilde{B}(m_l, u_j)$, where $\rho_j^{\texttt{REP}}$ is the length of reply packet for ψ_j.

Some steps may be skipped. If u_j stays in c_i and m_i has enough resources, we have $t_2 = t_4 = 0$, $m_k = m_i$ (in step 3), and $m_l = m_i$ (in step 5). Obviously, the response latency of task ψ_j is $\sum_{z=1}^{5} t_z$. If this latency exceeds τ_j (i.e., ψ_j's delay budget), ψ_j is viewed as *failed*. Then, our problem asks how to schedule tasks for cars (including task offloading) such that the *service ratio*, which is the ratio of non-failed tasks to the total tasks, can be maximized.

IV. The Proposed DTS Scheme

Algorithm 1 presents DTS's pseudocode. Suppose that a car u_j moves to a cell c_i (whose BMP is m_i), and u_j also issues a new task ψ_j. Line 2 estimates the number of RBs (denoted by $d_j^{\texttt{RB}}$) that m_i's BS shall allot to u_j to meet its communication rate λ_j. Due to page limits, we leave the detail of computing $d_j^{\texttt{RB}}$ in [21]. If the BS does not have enough RBs (i.e., line 3), u_j cannot be served. Otherwise, we update the BS's residual RBs (i.e., $r_i^{\texttt{RB}}$) by line 5. Then, we check if m_i's MEC server has enough CPU and MEM resources to handle u_j's task ψ_j. If so, we let m_i's MEC server process ψ_j and update its residual CPU and MEM resources (i.e., $r_i^{\texttt{CPU}}$ and $r_i^{\texttt{MEM}}$) accordingly. The code is given in lines 6–8.

Procedure `OffloadTask`($\hat{T}$, m_i):

1 $\Psi \leftarrow \emptyset$;
2 **foreach** $\psi_x \in \hat{T}$ **do**
3   **if** $\zeta_{i,x} \leq \zeta_{\texttt{th}}$ **then**
4         $\Psi \leftarrow \Psi \cup \{\psi_x\}$;
5 **if** $\Psi = \emptyset$ **then**
6   **foreach** $\psi_x \in \hat{T}$ **do**
7   **if** $\tau_x \geq \tau_{\texttt{th}}$ **then**
8             $\Psi \leftarrow \Psi \cup \{\psi_x\}$;
9 **return** Ψ;

Procedure `TargetBMP`(ψ_x, m_i):

1 **if** $\zeta_{i,x} \leq \zeta_{\texttt{th}}$ **then**
2   find u_x's target cell c_k;
3   **if** $r_k^{\texttt{CPU}} \geq d_x^{\texttt{CPU}}$ *and* $r_k^{\texttt{MEM}} \geq d_x^{\texttt{MEM}}$ **then**
4   **return** m_k;
5   **return** null;
6 $\hat{M} \leftarrow \emptyset$;
7 **foreach** $c_y \in \hat{C}_i^{\texttt{N}}$ **do**
8   **if** $r_y^{\texttt{CPU}} \geq d_x^{\texttt{CPU}}$ *and* $r_y^{\texttt{MEM}} \geq d_x^{\texttt{MEM}}$ **then**
9         $\hat{M} \leftarrow \hat{M} \cup \{m_y\}$;
10 **if** $\hat{M} = \emptyset$ **then**
11   **return** null;
12 Compute the score α_y for each BMP $m_y \in \hat{M}$;
13 **return** $\arg\max_{m_y \in \hat{M}} \alpha_y$;

Once m_i's MEC server has insufficient resources, some of its tasks (possibly including ψ_j) need to be offloaded to other MEC servers. To do so, we use two variables $r_i^{'\texttt{CPU}}$ and $r_i^{'\texttt{MEM}}$ to store m_i's residual CPU and MEM resources if m_i accepts ψ_j, as line 9 shows. Note that at least one of $r_i^{'\texttt{CPU}}$ and $r_i^{'\texttt{MEM}}$ is negative. Then, line 10 finds a subset Ψ_j of candidate tasks to be offloaded from the union of $\hat{T}_i$ (i.e., the set of m_i's current tasks) and ψ_j. This is done via the *OffloadTask* procedure. Line 11 sorts tasks in Ψ_j decreasingly by their sizes (i.e., s_j). For each task ψ_x in Ψ_j, we use the *TargetBMP* procedure to select a BMP m_k, where ψ_x will be offloaded to its MEC server later. Then, we mark ψ_x and increase $r_i^{'\texttt{CPU}}$ and $r_i^{'\texttt{MEM}}$ by $d_x^{\texttt{CPU}}$ and $d_x^{\texttt{MEM}}$ (i.e., ψ_x's demand for CPU and MEM resources). When both $r_i^{'\texttt{CPU}}$ and $r_i^{'\texttt{MEM}}$ become non-negative, m_i has enough resources to process tasks. Hence, we stop finding MEC servers to offload m_i's tasks (in Ψ_j). The code is shown in lines 12–18.

If $r_i^{'\texttt{CPU}}$ or $r_i^{'\texttt{MEM}}$ is still negative (i.e., line 19), m_i cannot have enough resources even if we offload all tasks in Ψ_j. Thus, ψ_j is dropped. Otherwise, we transfer each marked task ψ_x to its target BMP m_k and update m_k's residual resources in line 23. Afterward, line 24 sets $r_i^{\texttt{CPU}}$ and $r_i^{\texttt{MEM}}$ to $r_i^{'\texttt{CPU}}$ and $r_i^{'\texttt{MEM}}$, as the marked tasks are offloaded to other MEC servers.

A. The OffloadTask Procedure

Given a set $\hat{T}$ of tasks, this procedure helps a BMP m_i select a subset Ψ from $\hat{T}$ as candidates to offload, which considers two cases. For case 1, we choose the tasks whose requestors (i.e., cars) are in the cell edge [22]. The reason is that these cars are about to leave m_i's cell. Fig. 1 gives an example, where a car u_2 will leave cell c_3 soon. Thus, BMP m_3 can offload u_2's task ψ_2 to BMP m_2 (whose cell is u_2's handoff target). This way, when u_2 enters cell c_2, m_2 can directly reply to it (after ψ_2 is processed), thereby reducing the response latency. For case 2, there is no car in the cell edge. We pick those tasks with large delay budgets as they can tolerate longer response latency. For instance, the delay budgets of u_3's task (i.e., ψ_3) and u_4's task (i.e., ψ_4) are 100 ms and 300 ms in Fig. 1. Suppose that BMP m_1 chooses m_2 to offload a task, and u_3 and u_4 do not leave cell c_1. It is better to pick ψ_4 to offload, since ψ_4 could tolerate long latency for flow $u_4 \Rightarrow m_1 \Rightarrow m_2 \Rightarrow m_1 \Rightarrow u_4$.

In the pseudocode, lines 1–4 handle case 1, where $\zeta_{i,x}$ is the SINR between BMP m_i's BS and car u_x (whose task is ψ_x). If $\zeta_{i,x}$ is below threshold $\zeta_{\texttt{th}}$, u_x is in the cell edge. Thus, ψ_x is added to Ψ. However, if no task is found using case 1 (i.e., $\Psi = \emptyset$ in line 5), we pick tasks whose delay budgets (i.e., τ_x) are above threshold $\tau_{\texttt{th}}$ (i.e., case 2), as shown in lines 5–8.

B. The TargetBMP Procedure

This procedure selects an MEC server to offload a task ψ_x (issued by car u_x in cell c_i whose BMP is m_i). Lines 1–5 are for case 1 (i.e., u_x is in the cell edge). In line 2, we find u_x's target cell c_k. If c_k's MEC server has enough resources to serve ψ_x (i.e., line 3), we return m_k. Otherwise, ψ_x is not suitable to be offloaded, so line 5 returns null. Here, GIS can be used to find the target cell in line 2. If GIS is unavailable, we predict the target cell via vector calculation. Let $\vec{F}_x$ be the vector for u_x's moving direction and $\hat{C}_i^{\texttt{N}}$ be the set of cells adjacent to c_i. For each cell $c_y \in \hat{C}_i^{\texttt{N}}$, we build a vector $\vec{V}_{x,y}$ (from u_x to m_y) and calculate the cosine value of the angle θ between vectors $\vec{F}_x$ and $\vec{V}_{x,y}$ by $\cos\theta = (\vec{F}_x \cdot \vec{V}_{x,y})/(\|\vec{F}_x\| \cdot \|\vec{V}_{x,y}\|)$. A larger $\cos\theta$ value means that $\vec{F}_x$ and $\vec{V}_{x,y}$ are more similar. Hence, among all cells in $\hat{C}_i^{\texttt{N}}$, we pick the one with the largest $\cos\theta$ value. Fig. 1 gives an example, where vectors $\vec{F}_5$ and $\vec{V}_{5,7}$ are the most similar, so car u_5's target cell is c_7.

The residual code is for case 2. We use a set $\hat{M}$ to record adjacent cells in $\hat{C}_i^{\texttt{N}}$ whose MEC servers have enough resource to process task ψ_x. The code is in lines 6–9. If $\hat{M}$ is empty, line 11 returns null, as nearby MEC servers are all busy. Otherwise, for each BMP $m_y \in \hat{M}$, we compute a score α_y by

$$\frac{\omega_1\eta(r_k^{\texttt{CPU}}) + \omega_2\eta(r_k^{\texttt{MEM}}) + \omega_3\eta(\tilde{B}(m_i, m_y)) + \omega_4\eta(N_y)}{\sum_{z=1}^{4}\omega_z}, \quad (1)$$

where N_y is the number of m_y's tasks and $\eta(X)$ denotes a normalization function defined as $\eta(X) = (X - X_{\min})/(X_{\max} - X_{\min})$. As can be seen, Eq. (1) takes account of residual CPU and MEM resources of m_y, the bandwidth between m_i and m_y (which affects the task offloading time t_2 and the result passing time t_4), and the number of serving tasks. As discussed in Section III, the task handling time (i.e., t_3) rises as an MEC server handles more tasks. Thus, ω_4 is set to a negative value. Coefficients ω_1, ω_2, and ω_3 are set to positive values. Besides, we have $\sum_{z=1}^{3} \omega_z > \omega_4$. Finally, we pick the BMP with the highest score, as shown in line 13.

V. Performance Evaluation

In the simulation, SUMO [23] is adopted to model roads and car traffic. Specifically, we pick a 4 km $\times$ 4 km area $\mathcal{A}$ from the downtown of Kaohsiung, Taiwan, as shown in Fig. 2. The road map can be obtained using OpenStreetMap and then imported to SUMO via its road-network importer called *netconvert* [24]. Each road has a speed limit of 50 or 60 km/h. We place traffic lights on $\mathcal{A}$, whose cycles are 60 to 90 seconds. The maximum number of cars in $\mathcal{A}$ is 120 (i.e., $|\hat{U}| \leq 120$), where they move following the Manhattan grid model [25].

There are 18 cells deployed on $\mathcal{A}$ (i.e., $|\hat{C}| = 18$). As Fig. 2 shows, 16 cells (i.e., green ones) offer seamless coverage and 2 cells (i.e., red ones) are placed on hotspots. The cell range is 750 m. For a BS, the operating band, channel bandwidth, and transmission power is set to 2.6 GHz, 20 MHz, and 46 dBm, respectively. Each BS can provide 100 RBs/ms. The data rates for uplink (i.e., car $\Rightarrow$ BMP) and downlink (i.e., BMP $\Rightarrow$ car) are 180–270 Mbps and 300–380 Mbps. The data rate between two BMPs is 30–160 Mbps. As for channel fading (e.g., path loss, shadowing, and multipath fading), we follow the 3GPP specification for 5G, which can refer to [26] for more details.

Each MEC server has 50 CPU and 100 MEM resources. There are three kinds of tasks in terms of resource demands. Small-demand tasks request for $[1, 3]$ CPU and $[1, 6]$ MEM resources, medium-demand tasks require $[4, 7]$ CPU and $[7, 16]$ MEM resources, and large-demand tasks need $[8, 12]$ CPU and $[16, 19]$ MEM resources. The proportion of tasks of each kind is equal (i.e., 1/3 of total tasks). Besides, there are three delay budgets: 100 ms (low), 150 ms (medium), and 300 ms (high). Let us consider three scenarios. In scenarios Q1, Q2, and Q3, the ratios of tasks with low, medium, and high delay budgets are set to 10:40:50, 15:35:50, and 20:30:50. Each car issues a task every 200 s to 300 s. The simulation time is 4000 s. Thus, there will be about 1800 tasks in total.

Two methods are for comparison. The RSRP-based method [27] selects the MEC server whose partner BS has the maximum RSRP (reference signal received power) for task offloading. The TOPSIS-based method [17] chooses MEC servers to offload tasks by referring to their resources and bandwidth. For our DTS scheme, we evaluate performance when using GIS and vector calculation to predict cars' target cells. Moreover, we set $\tau_{\texttt{th}} = 150$ ms, $\omega_1 = 1$, $\omega_2 = 1$, $\omega_3 = 3$, and $\omega_4 = -3$.

Fig. 3(a) gives the average service ratio. This ratio reduces as the proportion of tasks with low delay budgets (i.e., 100 ms) rises. That is why each method has the lowest service ratio in scenario Q3. The RSRP-based method chooses BMPs whose BSs have better signal quality to offload tasks, but the MEC servers of some chosen BMPs may have insufficient resources. This makes the service ratio in the RSRP-based method below 60%. By considering resources of MEC servers, the TOPSIS method can increase the service ratio to 82%–85%. Our DTS scheme adaptively transfers tasks with high delay budgets to nearby MEC servers based on their resources, bandwidth, and serving tasks. Hence, DTS significantly improves the service ratio. More concretely, the service ratio in our DTS scheme can be higher than 95% and 91% when using GIS and vector calculation to predict handoff targets of cars, respectively.

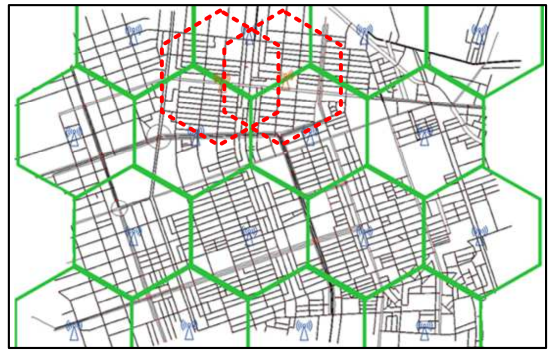


Fig. 2. Road map (Kaohsiung) and cell deployment in the simulation.

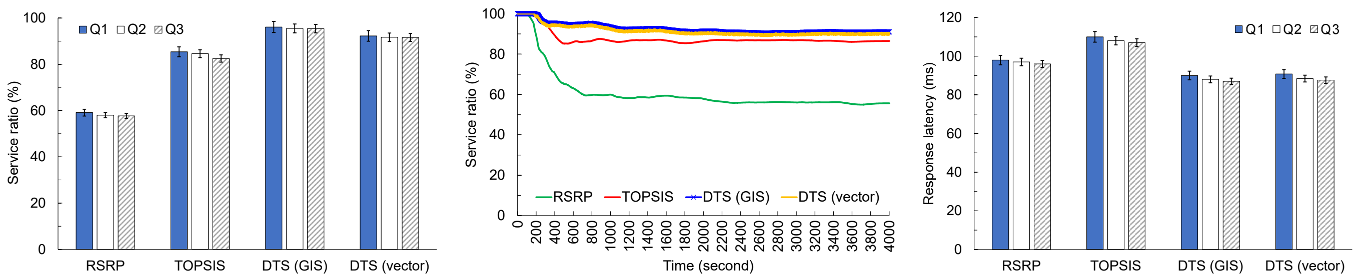
Fig. 3(b) shows the changes of service ratio over time (in scenario Q3). Initially, only a few cars enter area $\mathcal{A}$, so BMPs have enough resources to serve their tasks. As the number of cars increases, some MEC servers become busy, so the service ratio of each method decreases. This phenomenon is especially obvious in the RSRP-based method. After 700 s, the number of cars in $\mathcal{A}$ is stabilizing (and reaches the maximum value), so the service ratio of each method becomes stable accordingly.

Fig. 3(c) gives the average response latency for non-failed tasks. The RSRP-based method selects BMPs whose BSs have large RSRP values for task offloading. In this case, there is a good possibility that cars will move to these cells. Doing so may reduce the time taken to the result passing step (i.e., t_4). That explains why the RSRP-based method has lower response latency than the TOPSIS-based method. Our DTS scheme finds BMPs to offload tasks by referring to cars' moving directions. Besides, DTS takes account of the number of tasks served by MEC servers, which saves the time spent by the task handling step (i.e., t_3). Thus, our DTS scheme has the lowest response latency among all methods.

Then, we measure the accuracy of predicting target cells for cars in our DTS scheme. In particular, when using GIS and vector calculation for prediction, the accuracy is 92.3% and 88.6%. As can be seen, the gap is not large (below 4%). This can explain why the performance difference between the DTS method using GIS and vector calculation is small in Fig. 3.

VI. Conclusion

This paper proposes the DTS scheme to schedule tasks for MEC servers in an IoV environment. When a car moves into



(a) average service ratio (b) service ratio over time (Q3 scenario) (c) response latency

Fig. 3. Performance comparison of different methods.

a cell, its MEC server may not possess enough resources to process the car's task. Therefore, DTS refers to cars' locations and tasks' delay budgets to offload parts of tasks of the MEC server. If a car is about to move to another cell, the car's task is offloaded to the MEC server of the target cell. Otherwise, DTS chooses nearby MEC servers for offloading based on residual resources, bandwidth, and the number of serving tasks. If GIS is not available, we predict target cells for cars through vector calculation. Through simulations using SUMO, we show that our DTS scheme can efficiently improve the service ratio while reducing the response latency compared with RSRP-based and TOPSIS-based methods.

Acknowledgment

This work was supported by National Science and Technology Council, Taiwan under Grant 111-2221-E-110-023-MY2.

References

[1] W. Zhuang, Q. Ye, F. Lyu, N. Cheng, and J. Ren, "SDN/NFV-empowered future IoV with enhanced communication, computing, and caching," *Proceedings of the IEEE*, vol. 108, no. 2, pp. 274–291, 2020.

[2] Y. C. Wang and C. E. Ho, "Speed-aware flow management with packet classification to mitigate congestion in VANETs," in *IEEE VTS Asia Pacific Wireless Communications Symposium*, 2023, pp. 1–5.

[3] G. Yan, K. Liu, C. Liu, and J. Zhang, "Edge intelligence for Internet of vehicles: A survey," *IEEE Trans. Consumer Electronics*, early access, Mar. 2024, doi: 10.1109/TCE.2024.3378509.

[4] P. Ranaweera, A. D. Jurcut, and M. Liyanage, "Survey on multi-access edge computing security and privacy," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1078–1124, 2021.

[5] C. Campolo, A. Iera, A. Molinaro, and G. Ruggeri, "MEC support for 5G-V2X use cases through Docker containers," in *IEEE Wireless Communications and Networking Conference*, 2019, pp. 1–6.

[6] S. D. A. Shah, M. A. Gregory, S. Li, and R. D. R. Fontes, "SDN enhanced multi-access edge computing (MEC) for E2E mobility and QoS management," *IEEE Access*, vol. 8, pp. 77 459–77 469, 2020.

[7] N. Monir *et al.*, "Seamless handover scheme for MEC/SDN-based vehicular networks," *Journal of Sensor and Actuator Networks*, vol. 11, no. 1, pp. 1–16, 2022.

[8] P. A. Frangoudis and A. Ksentini, "Service migration versus service replication in multi-access edge computing," in *International Wireless Communications & Mobile Computing Conference*, 2018, pp. 124–129.

[9] K. S. Hsu, W. C. Chang, W. H. Huang, and P. C. Wang, "Adaptive replication for real-time applications based on mobile edge computing," in *IEEE International Conference on Communication, Networks and Satellite*, 2021, pp. 88–94.

[10] S. A. Mohamed, S. Sorour, and H. S. Hassanein, "Group delay-aware scalable mobile edge computing using service replication," *IEEE Trans. Vehicular Technology*, vol. 71, no. 11, pp. 11 911–11 920, 2022.

[11] P. Dai *et al.*, "Distributed convex relaxation for heterogeneous task replication in mobile edge computing," *IEEE Trans. Mobile Computing*, vol. 23, no. 2, pp. 1230–1245, 2024.

[12] I. Labriji *et al.*, "Mobility aware and dynamic migration of MEC services for the Internet of vehicles," *IEEE Trans. Network and Service Management*, vol. 18, no. 1, pp. 570–584, 2021.

[13] W. Chen *et al.*, "MSM: Mobility-aware service migration for seamless provision: A data-driven approach," *IEEE Internet of Things Journal*, vol. 10, no. 17, pp. 15 690–15 704, 2023.

[14] F. Li, Y. Lin, N. Peng, and Y. Zhang, "Deep reinforcement learning based computing offloading for MEC-assisted heterogeneous vehicular networks," in *IEEE International Conference on Communication Technology*, 2020, pp. 927–932.

[15] X. Liu, C. Zhang, and S. He, "Adaptive task offloading for mobile aware applications based on deep reinforcement learning," in *IEEE International Conference on Mobile Ad Hoc and Smart Systems*, 2022, pp. 33–39.

[16] T. D. T. Nguyen *et al.*, "Modeling data redundancy and cost-aware task allocation in MEC-enabled Internet-of-vehicles applications," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1687–1701, 2021.

[17] D. Patil and E. Al-Masri, "Seamless service migration across multi-access edge computing (MEC) environments," in *IEEE Eurasia Conference on IoT, Communication and Engineering*, 2021, pp. 369–375.

[18] Y. C. Wang and J. F. Zhang, "ERA: Efficient request assignment for servers in data center networks with SDN," in *IEEE International Conference on Smart Communities: Improving Quality of Life Using AI, Robotics and IoT*, 2023, pp. 24–29.

[19] Y. Li, C. Yang, M. Deng, X. Tang, and W. Li, "A dynamic resource optimization scheme for MEC task offloading based on policy gradient," in *IEEE Information Technology and Mechatronics Engineering Conference*, 2022, pp. 342–345.

[20] W. K. Lai, Y. C. Wang, and S. C. Wei, "Delay-aware container scheduling in Kubernetes," *IEEE Internet of Things Journal*, vol. 10, no. 13, pp. 11 813–11 824, 2023.

[21] Y. C. Wang and K. C. Chien, "EPS: Energy-efficient pricing and resource scheduling in LTE-A heterogeneous networks," *IEEE Trans. Vehicular Technology*, vol. 67, no. 9, pp. 8832–8845, 2018.

[22] Y. C. Wang and D. R. Jhong, "Efficient allocation of LTE downlink spectral resource to improve fairness and throughput," *International Journal of Communication Systems*, vol. 30, no. 14, pp. 1–13, 2017.

[23] SUMO, https://www.eclipse.org/sumo/.

[24] Y. C. Wang and G. W. Chen, "Efficient data gathering and estimation for metropolitan air quality monitoring by using vehicular sensor networks," *IEEE Trans. Vehicular Technology*, vol. 66, no. 8, pp. 7234–7248, 2017.

[25] W. H. Yang, Y. C. Wang, Y. C. Tseng, and B. S. P. Lin, "Energy-efficient network selection with mobility pattern awareness in an integrated WiMAX and WiFi network," *International Journal on Communication Systems*, vol. 23, no. 2, pp. 213–230, 2010.

[26] Y. C. Wang and C. W. Chou, "Efficient coordination of radio frames to mitigate cross-link interference in 5G D-TDD systems," *Computer Networks*, vol. 232, pp. 1–13, 2023.

[27] S. Wu, J. Ren, T. Zhao, and Y. Wang, "Machine learning based signal strength and uncertainty prediction for MEC mobility management," in *IEEE Vehicular Technology Conference*, 2021, pp. 1–5.