Digital Twin Empowered Task Offloading for Vehicular Edge Computing

Chaogang Tang*, Huaming Wu†, Chunsheng Zhu‡, Shuo Xiao*
*School of Computer Science and Technology, China University of Mining and Technology, 221116, Xuzhou, China
† Center for Applied Mathematics, Tianjin University, 300072, Tianjin, China
‡College of Big Data and Internet, Shenzhen Technology University, 518118, Shenzhen, Guangdong, China
cg.tang@foxmail.com, whming@tju.edu.cn, chunsheng.tom.zhu@gmail.com, sxiao@cumt.edu.cn

***Abstract*—Vehicular edge computing (VEC) as a promising computing paradigm has accelerated the reformation of existing dominating computing infrastructures, enabling resource provisioning in close proximity to resource requestors. However, several challenges still exist, including efficient resource scheduling and management, dynamic wireless channel state, and limited bandwidth usage. To address these issues, we introduce the digital twin (DT) technology into VEC, enabling DTs of physical entities in VEC to achieve real-time offloading decision-making in the DT simulation cycle. In particular, we propose a DT-empowered VEC (DT-VEC) architecture, aiming to achieve efficient task offloading while considering extra latency incurred by task migration. We further put forward an efficient algorithm to minimize the response latency for all the tasks in the optimization period. The simulation results have proven that our approach outperforms the other two greedy approaches.**

***Index Terms*—Vehicular edge computing, task offloading, digital twin, response latency, resource scheduling**

I. Introduction

Vehicular edge computing (VEC) has emerged as a promising computing paradigm that has expedited the reformation of existing dominating computing infrastructures for vehicles, such as vehicular cloud computing [1]. VEC enables computing resources provisioning for vehicular tasks and applications in a more scalable and adaptive way, by pushing the computing and caching resources of the cloud center into the logical edge of vehicular networks, such as roadside units (RSU) deployed along the road. The benefits of VEC are at least threefold. Firstly, it significantly fulfills the real-time requirement of a variety of vehicular tasks, including online in-car games, navigation-related image rendering tasks, vehicular media applications, etc. Secondly, VEC can tremendously relieve the computational pressure of the on-board units (OBU) of vehicles, particularly in light of the explosive growth of vehicular applications. Thirdly, through the deployment of lightweight edge servers at RSUs located in close proximity to the resource requestors (e.g., vehicles), VEC ensures a high level of quality of service (QoS) during the service and resource provisioning process, thereby guaranteeing reliable and timely delivery of services and resources to the resource requestors. Additionally, VEC enhances the quality of experience (QoE) for resource requestors, ensuring that the users' expectations and satisfaction are met by providing a seamless and uninterrupted service experience [2].

Despite the aforementioned advantages, the following challenges revolving around task offloading in VEC are still not well addressed. The computing and caching capabilities of RSUs are much weaker than the cloud center, which requires more efficient and effective service provisioning and resource management in VEC. Furthermore, the high mobility of vehicles can not only incur dynamics of vehicular ad hoc networks (VANET), but also require task migration among adjacent RSUs. For instance, a vehicle may not receive the feedback results before leaving the coverage of the current serving RSU to which its task is offloaded. In this case, task migration is required, which brings about extra migration costs on caching, networking and computing power. Last, both wireless channel state and bandwidth can contribute to the difficulty in efficiently designing task offloading strategies and computing resource management schemes in VEC.

The digital twin (DT) technology can offer a promising solution to the above issues in VEC. DT can be defined as a virtual model to depict a physical asset by data-driven simulators [3], [4]. DT is initially introduced to intelligent manufacturing for real-time controlling, monitoring and optimizing [5]. Similarly, the DT can benefit VEC by undertaking decision-making roles in the DT simulation cycle. In particular, DT can efficiently mirror the physical entities such as the behaviors and topological changes, by analyzing and predicting the states of these entities in real-time [6].

In this paper, we put forward a DT-empowered VEC (DT-VEC) architecture, with the aim to achieve efficient task offloading, and further improving the QoS from the perspective of resource providers (i.e., RSUs or edge servers) and the QoE from the perspective of resource requestors (i.e., vehicles with task offloading requests). The contributions in this paper are threefold.

- We propose a DT-VEC architecture for task offloading and resource allocation. This architecture comprises two layers, namely the physical layer and the DT layer. Specifically, the DT layer is composed of digital replicas of both RSUs and the entire VEC system.
- An efficient task offloading algorithm is proposed in this

paper to minimize the total response latency for all tasks within the optimization period. Different from existing works, the latency incurred by task migration is also considered in this paper.
- Extensive simulation experiments were conducted to evaluate our approach in comparison with two other existing greedy approaches. The results demonstrate that our approach achieves a significant improvement in response latency optimization, demonstrating its effectiveness and efficiency.

II. Related Works

Many fields have adopted DT technologies to realize real-time controlling, monitoring and optimizing. Current works, in the context of new computing paradigms, try to investigate and discuss the roles and responsibilities of DTs which can undertake in various application scenarios.

Considering many works in mobile edge computing (MEC) that ignore the effects of user mobility and environments, a DT-enabled edge network was proposed in [3]. In this architecture, DTs play multiple roles, including the evaluation of edge servers and the entire MEC system. They attempted to optimize the response latency using Actor-Critic deep reinforcement learning and obtain better performance in the simulation. Schwarz *et al.* [7] investigated the use of DT in connected and automated vehicles (CAVs). They stated that DTs should have direct communication with their counterparts in physical systems. Therefore, new methods that adopt DT technologies are required for testing CAVs efficiently, as shown in their review of vehicular DTs. They also discussed open gaps and challenges for the sustainable development of DT applications.

To improve the efficiency of intelligent transportation, Schwarz *et al.* [8] have explored the combination of blockchain with DTs in vehicle management. In particular, considering the difficulties in predicting pedestrians in reality, they use DTs to mirror the traffic situation on real roads. In addition, the blockchain technology is used to secure the interaction of vehicle data information. They further construct the DTs of vehicle Ad Hoc networks based on blockchain technology. Lv *et al.* [9] combined DT with Artificial Intelligence (AI) to prevent traffic accidents and guarantee the safety of drivers and pedestrians. They used the Long-Short Term Memory (LSTM) network to predict the low-frequency sub-layers. They further built a double-scale LSTM network prediction model, which it has a strong power in predicting traffic accidents. Chukhno *et al.* [10] attempted to address the issue of DTs placement at the edge, considering the IoT device mobility, features of edge network and the corresponding social peculiarities. They put forward an approximation algorithm to solve the formulated quadratic assignment problem. Their heuristic rules guarantee that the proposed algorithm can almost approach the optimal solution at a low time complexity. Lv *et al.* investigated the storage security of edge-fog-cloud computing. To this end, the DT technology is used to simulate the online data-driven behaviors of machine manufacturing. Furthermore, deep learning technology is used to prevent network intrusion. The simulation results show that their approach has better performance than other approaches. They are also other works such as [11]–[13] which focus on the applications of DT technologies. Owing to space limitations, we do not detail them anymore.

III. System Model

A. Architecture for Digital Twin Empowered Vehicular Edge Computing

The architecture of DT-VEC for task offloading and resource allocation is presented in Fig. 1, which is composed of two layers, namely the physical layer and the DT layer. The physical layer includes the entities involved in the VEC networks, such as smart vehicles, edge server-enhanced RSUs, and a cloud center. Vehicles communicate with each other and RSUs via vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) wireless links, respectively. RSUs are inter-connected with each other using wired links such as high-capacity optical fibers. A cloud center in this physical layer is responsible for supplementing computing and caching resources for RSUs, since the cloud center is much richer in these resources than RSUs.

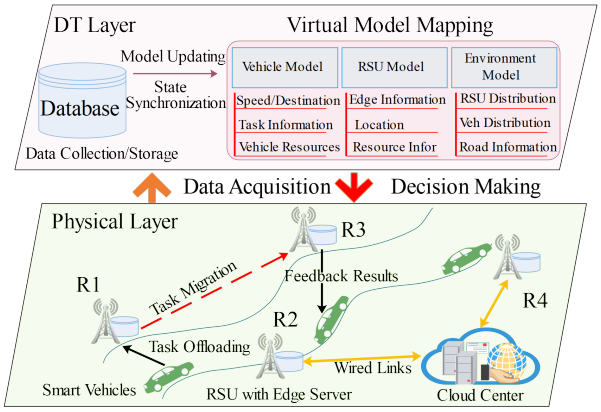

Fig. 1. The architecture of DT-VEC for task offloading and resource allocation.

The proposed DT layer comprises digital replicas of the physical entities in VEC, with a focus on both individual DTs of RSUs and the global DT of the entire VEC system. These DTs are responsible for monitoring and analyzing the states of physical entities in VEC and making efficient offloading decisions. For instance, the DTs of RSUs interact with passing vehicles through V2I communication links to gather and record information such as destination, velocity, and offloading requests. Moreover, the DTs of RSUs periodically update their records of available computing, caching, and networking resources. It is worth noting that these DTs are located either

on the corresponding RSUs or on other RSUs with abundant resources.

In the above architecture, we consider one moving vehicle that has different task offloading requests in the optimization period $\mathcal{T}$. The optimization period is divided into discrete time slots $\mathcal{T} = \{1, \cdots, T\}$ where T is the number of time slots. Denote by $\mathcal{R} = \{R_1, \cdots, R_M\}$ a set of RSUs which are deployed along the road and M is the number of RSUs. The DT of RSU m can be calculated as:

$$DT_m = \Theta(f_m, \Delta f_m), \tag{1}$$

where f_m is the estimated computing capability of RSU m, and Δf_m is an estimated deviation from the true computing capability of m, since the DTs cannot accurately image the state of RSUs and thus a deviation always exists.

To simplify the discussion, we assume that the vehicle v has only one task to offload in each time slot. Denote by $B(t) = (\zeta(t), \chi(t), \gamma(t))$ the task generated by v in time slot t, where $\zeta(t)$ denotes the task-input size, $\chi(t)$ is the number of computing resources required to accomplish $B(t)$, and $\gamma(t)$ is the deadline of the response latency for $B(t)$. As shown in the figure, vehicle v may be covered by multiple RSUs at the same time, since RSUs can be densely deployed along the road. Thus, denote by $\mathcal{C}(t)$ the set of RSUs that cover v in time slot t. In this case, an arbitrary RSU $R_m \in \mathcal{C}(t)(\mathcal{C}(t) \subseteq \mathcal{R})$ can be a potential candidate to which $B(t)$ can be offloaded. Let $K(t)$ denote the dwelling time of v within the coverage of the serving RSU in time slot t, and $K(t)$ can be estimated based on the coverage area of RSU and the velocity of the vehicle [14].

B. Task Offloading Model

Offloading $B(t)$ to the corresponding serving RSU R_m causes the offloading delay, and retrieving the execution results also causes the feedback delay. In this paper, we assume that v adopts an orthogonal spectrum for task offloading to get rid of the co-channel interference. Thus, the data rate for task offloading can be calculated as:

$$r(t) = W \log\left(1 + \frac{P(t)G_m(t)}{\delta^2}\right), \tag{2}$$

where W, $P(t)$ and $G_m(t)$ are the uplink channel bandwidth between v and RSU, transmission power of v in slot t, and channel gain between v and RSU R_m in slot t, respectively. δ^2 denotes the noise power. Thus, the offloading delay denoted by $T^m_{off}(t)$ is:

$$T^m_{off}(t) = \frac{\zeta(t)}{r(t)}. \tag{3}$$

C. Computation Model

When $B(t)$ is offloaded to R_m, the edge server deployed at the RSU will create a virtual machine and allocate the computing resources for accomplishing it. In the DT-VEC architecture, the state of the edge server such as its processing frequency can be estimated by its DT. In this case, the estimated computation latency $\widetilde{T}^m_{cmp}(t)$ for the task in time slot t can be calculated as:

$$\widetilde{T}^m_{cmp}(t) = \frac{\chi(t)}{f_m}. \tag{4}$$

Owing to the existence of an estimated deviation from the true computing capability of RSU, we also need to consider the computation latency gap during evaluating the computation latency. The computation latency gap, denoted by $G^m_{cmp}(t)$, can be expressed as [3]:

$$G^m_{cmp}(t) = -\frac{\chi(t)\Delta f_m}{f_m(f_m + \Delta f_m)}. \tag{5}$$

Accordingly, the actual computation latency for the task performed at R_m in time slot t can be expressed as:

$$T^m_{cmp}(t) = \widetilde{T}^m_{cmp}(t) + G^m_{cmp}(t). \tag{6}$$

Note that the execution results generated by computation-oriented task offloading and execution at RSUs usually need to be sent back to the moving vehicles in VEC. However, to simplify the discussion, many works assume that the data size of the execution result is much smaller than the task-input data size, and thus the feedback delay can be ignored [15], [16]. Accordingly, in this paper, we also ignore the feedback delay, as assumed in these works.

D. Task Migration Model

In this paper, we assume that all tasks must be completed. However, owing to the stringent latency requirements and high mobility of smart vehicles, the execution result for a vehicular task may not be delivered to the vehicle on time before it leaves the coverage of the RSU. In such cases, task migration is required to successfully accomplish the task. Task migration occurs at two levels, i.e., the source code level and computation result level, respectively. Particularly, the former occurs when a vehicle, which is moving away from the serving RSU but has not received the feedback result, re-offloads the unfinished task to the RSU it is approaching. The latter occurs when a vehicle, which is moving away from the current RSU but has not received the feedback result, does not re-offload the unfinished task to the RSU it is approaching but directly receives the feedback result from another RSU responsible for forwarding the result from the serving RSU. Both ways require the involvement of other RSUs, which act as either the computing node or the forwarding node in the process of task migration. The computation result level is usually preferred over the source code level because it does not recalculate the offloading delay or computation delay.

Nevertheless, the task migration at the computation result level still incurs the migration latency in the DT-VEC system such as latency on re-authentication and result delivery in the physical layer and latency on state updates for RSUs in the DT layer. To ease the discussion, let C denote the average transmission delay for the task migration between two RSUs.

As a result, the latency of task migration for $B(t)$ selecting R_m as the serving edge in time slot t, denoted by $T^m_{mgr}(t)$, can be calculated as

$$T^m_{mgr}(t) = C \cdot \mathbb{F}\{T^m_{off}(t) + T^m_{cmp}(t) - K(t)\}, \quad (7)$$

where the function $\mathbb{F}\{\cdot\}$ is a unit-step function, i.e., $\mathbb{F}\{x\} = 1$ for $x \geq 0$, and 0, otherwise. We use this function to denote the decision for the task migration. For instance, if the sum of the offloading delay and computation delay is larger than the estimated dwelling time of v within the coverage of the serving RSU, the execution result cannot be delivered on time to v, since v has moved far away from the serving RSU. In this case, the task migration is required.

Accordingly, the total response latency for $B(t)$ executed at R_m can be expressed as:

$$T^m_{rl}(t) = T^m_{cmp}(t) + T^m_{cmp}(t) + T^m_{mgr}(t). \quad (8)$$

IV. Problem Formulation

Define a binary variable $x_m(t)$ to denote whether the task generated in time slot t (i.e., $B(t)$) is offloaded to R_m. $x_m(t) = 1$ indicates that $B(t)$ is offloaded to R_m, and $x_m(t) = 0$ indicates that $B(t)$ is not offloaded to R_m.

Define the decision variable $\boldsymbol{x}_m = \{x_m(t)|t = 1, \cdots, T\}, \forall m \in \{1, \cdots, M\}$ to represent the offloading decision for R_m along the entire optimization period. Further define the decision profile $\boldsymbol{x} = \{\boldsymbol{x}_m | m = 1, \cdots, M\}$ for all the RSUs along the entire optimization period.

Then, the goal of this paper is to minimize the response latency for the generated tasks by v along the optimization period with the assistance of DTs. Specifically, the optimization problem can be formulated as follows:

$$\begin{aligned}
\boldsymbol{\mathcal{P}}: \ & \min_{\boldsymbol{x}} \quad \sum_{t=1}^{T} \sum_{m=1}^{M} x_m(t) \cdot T^m_{rl}(t), \\
s.t. \ & \sum_{m=1}^{M} x_m(t) = 1, \quad \forall t \in \mathcal{T}, && (9) \\
& T^m_{rl}(t) \leq \gamma(t), \quad \forall t \in \mathcal{T}, && (10) \\
& R_m|_{x_m(t)=1} \in \mathcal{C}(t), \quad \forall t \in \mathcal{T}, && (11) \\
& f_m \leq f^{max}_m, \quad \forall m \in \{1, \cdots, M\}, && (12) \\
& x_m(t) \in \{0, 1\}, \quad \forall m \in \{1, \cdots, M\}, \forall t \in \mathcal{T}, && (13)
\end{aligned}$$

where Eq. (9) ensures that the task $B(t)$ can be offloaded to only one RSU for execution at the same time in each time slot, which also implies that the task is indivisible in this paper. However, from the cost-efficient perspective, we allow for task migration at the computation result level in this paper. Eq. (10) means that the deadlines for tasks generated in the optimization period should be strictly satisfied, when making the offloading decisions in each time slot. Eq. (11) guarantees that the chosen RSU to serve the offloading request must cover v for the consequent V2I communications. We also assume that the computing resources allocated to the offloaded tasks by the serving RSU are bounded, since the remaining computing resources can be kept for other purposes. This can be realized by Eq. (12).

V. Algorithm Design

In this paper, we aim to minimize the total response latency for all the tasks along the optimization period by task offloading and computing resource allocation when the vehicle is travelling on the road. Actually, it is difficult to obtain the optimal solution in reality, owing to the following reasons. First, the vehicle v has no global information on the surrounding RSUs (i.e., the edge servers). Frequent interactions between v and RSUs such as beacon information exchanging not only incur extra response latency, but also consumes more energy and networking resources. Second, for security reasons, RSUs may not be willing to disclose their information such as their computing capabilities. Such factors may increase the difficulty in the offloading decision-making by v. Last but not least, realizing optimal task scheduling and possible task migration for vehicular tasks usually requires a centralized entity that is responsible for information gathering and decision-making. Considering the independence and autonomy of RSUs, it may be challenging to select one leading RSU from them.

In the DT-VEC architecture proposed in this paper, we can realize almost real-time task offloading and migration by means of DT technology. For instance, the DT layer can determine which RSU the task is offloaded to, and which RSU the task is migrated to if the task is not executed successfully. Note that the evaluation for the processing frequency requires the assistance of the DT technology. On the one hand, each DT of RSUs updates its states. Considering the fact that RSUs usually keep certain computing resources for other purposes such as running the individual DTs and global DT, the number of computing resources used for the vehicular task varies dynamically in each time slot. Thus, it is necessary for each DT of RSUs to evaluate the computing resources available in each time slot. On the other hand, the global DT can reflect the state of the system by gathering various information from individual DTs. Thus, it can make globally optimal decisions on task offloading and migration.

As a matter of fact, various algorithms can be employed to achieve efficient decision-making on task offloading and resource allocation in DT-VEC. In this paper, we present a novel and efficient algorithm with low-time complexity to solve problem P. We aim to ensure that the time spent on decision-making greatly caters to the strict response latency requirement for vehicular tasks. The details of the proposed algorithm are provided in Alg. 1. In particular, the procedure of the algorithm can be sketched out as follows:

Step 1 : At the beginning of each time slot, task $B(t)$ is generated by v. v communicates with the surroundings (e.g., RSUs via wireless V2I links). On the other hand, each individual DT of RSU monitors their states such as f_m and Δf_m. For instance, they

evaluate the maximal allowed amount of computing resources to accomplish the task in the current time slot.

Step 2 : The global DT gathers information from individual DTs. Specifically, a set of RSUs $\mathcal{C}(t)$ is built based on the information (line 5). An arbitrary RSU from $\mathcal{C}(t)$ can be a candidate RSU that undertakes the computation of $B(t)$. The algorithm checks each RSU R_m in $\mathcal{C}(t)$ and calculates the corresponding response latency, assuming that the task $B(t)$ is allocated to R_m (lines 6–8).

Step 3 : The values of $T_{rl}^m(t)$ $m \in \{1, \cdots, M\}$ are recorded. The algorithm selects the RSU R_m that has the minimal value of $T_{rl}^m(t)$, $m \in \{1, \cdots, M\}$ (line 10). Then, we check if there is a constraint violation if $B(t)$ is allocated to R_m (lines 11–15). If there is no constraint violation, the allocation is feasible. Otherwise, we ignore (e.g., remove) R_m from $\mathcal{C}(t)$, and continue this process (lines 9–16) until we find a feasible task allocation scheme. Then the algorithm makes per-slot task offloading decisions by setting $x_m(t) = 1$.

Step 4 : At the end, the algorithm will update the task offloading decision profile $\boldsymbol{x}$. Given $\boldsymbol{x}$, the algorithm will calculate the response latency for all the tasks along the optimization period.

VI. Performance Evaluation

In this section, we evaluate our approach by conducting a series of experiments in DT-VEC. The simulation parameters can be set as follows. The task $B(t)$, $t \in \mathcal{T}$ at each time slot is randomly generated by v that moves at a constant speed along a straight road of the length 10km. The number of RSUs deployed along the road varies between 20 and 50, and the communication range of each RSU ranges from 500m to 1000m. Based on the information, we can estimate the dwell time of vehicle v within the coverage of each RSU.

Intuitively speaking, one RSU is very attractive and preferred, if it has more computing resources and longer dwelling time for the passing vehicle. This observation introduces two greedy rules. That is to say, one rule tends to select the RSU that has the most powerful computing capacity as the potential offloading destination, and the other rule tends to select the RSU that has the longest dwell time for the vehicle as the potential offloading destination. Accordingly, we adopt the two greedy heuristic algorithms as benchmarks to compare with our approach in terms of efficiency and effectiveness. For simplicity, we call the two algorithms "Processing-optimal" and "Time-optimal", respectively.

Fig. 2 presents a comparison of the performance of the three approaches when the number of tasks is relatively small. Several conclusions can be drawn from the figure. Firstly, our approach outperforms the other two approaches regardless of the number of tasks. It is worth noting that the number of tasks

Algorithm 1: Efficient task offloading algorithm in DT-VEC (ETOA)

Input: Involved parameters in each time slot, such as $B(t)$, $K(t)$, $\mathcal{C}(t)$

Output: The optimal value for the response latency

1 **for** $t = 1$ to T **do**
2 Each DT_m evaluates and obtains f_m and Δf_m;
  `// Individual DTs can execute above codes in parallel`
3 Global DT gathers information from individual DTs;
  `// Global DT executes the following codes`
4 Initialize $x_m(t) = 0, \forall m \in \{1, \cdots, M\}$;
5 Determine $\mathcal{C}(t)$ by V2I communications;
6 **for** each R_m in $\mathcal{C}(t)$ **do**
7 Given $B(t)$, calculate $T_{rl}^m(t)$ according to Eq. 8;
8 **end**
9 **while** task offloading decision not made **do**
10 Select one R_m with the minimal value of $T_{rl}^m(t)$ currently;
11 **if** Constraints not violated **then**
12 Select R_m as the offloading destination;
13 **else**
14 Ignore R_m in $\mathcal{C}(t)$;
15 **end**
16 **end**
17 Allocate $B(t)$ to R_m and set $x_m(t) = 1$;
18 **end**
19 Update $\boldsymbol{x}$;
20 Calculate and return the response latency based on $\boldsymbol{x}$;

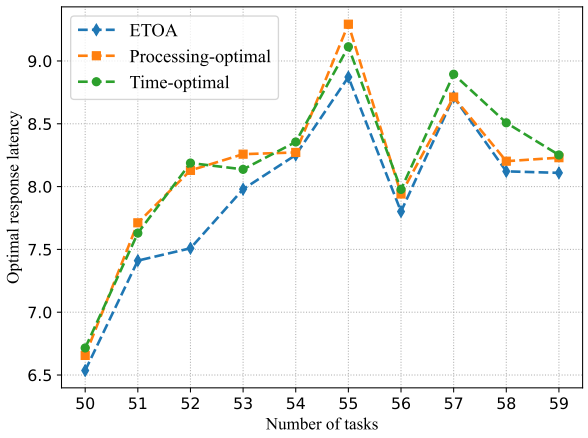

Fig. 2. Performance comparison with a small number of vehicular tasks.

is assumed to be equal to the number of time slots in this paper, and only one task is generated by v in each time slot. Secondly, there is no consistent pattern between "Processing-optimal" and "Time-optimal." Sometimes the former outperforms the

latter, and sometimes it does not. Since latency minimization is the performance metric, the computing power of RSUs is directly related to the response latency of vehicular tasks. In most cases, the former performs better than the latter, as also shown in this figure. Thirdly, since the tasks of v and the computing resources of RSUs are randomly generated during the optimization period, the minimal response latency varies significantly across time slots.

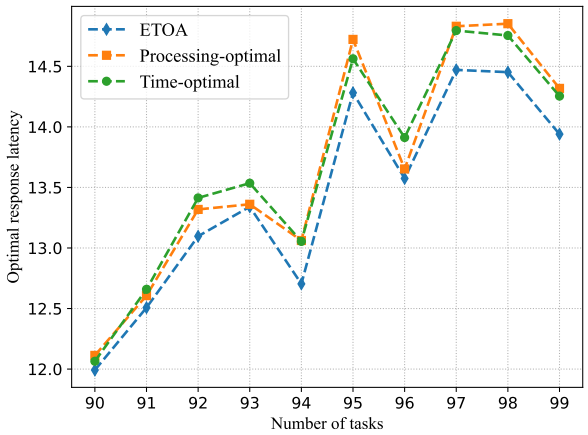

Fig. 3. Performance comparison with a large number of vehicular tasks.

Fig. 3 compares the performance of the three approaches when the number of tasks is relatively large (e.g., ranging from 90 to 100). The observations drawn from this figure are consistent with those made in Fig. 2, and therefore, we do not reiterate them. The combination of the insights gleaned from the two figures leads us to conclude that our approach outperforms the two greedy approaches, irrespective of the number of tasks being small or large.

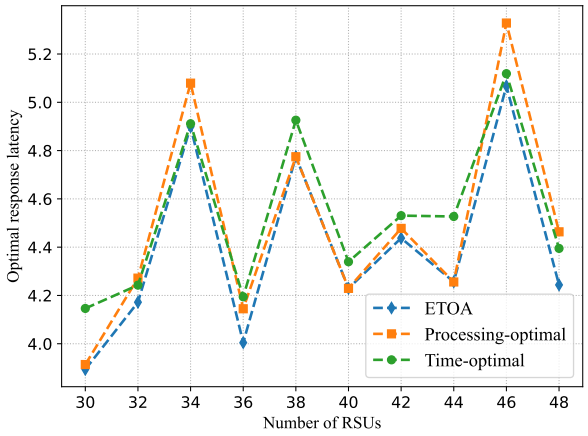

Fig. 4. Performance comparison with the increasing number of RSUs.

Fig. 4 illustrates a performance comparison among the three approaches when the number of RSUs varies between 30 and 50. Several observations can be made from the results. Firstly, our proposed approach outperforms the other two approaches across different numbers of RSUs. Moreover, the total response latency does not decrease with the increasing number of RSUs, which may contradict regular expectations. There are several reasons for this interesting observation. For instance, all the information related to task and edge servers deployed at RSUs is generated randomly in each time slot. Thus, the increase in the number of RSUs does not necessarily lead to an increase in computing resources. Additionally, other factors such as the difference in the dwelling time of RSUs covering v can impact the response latency for all the tasks. Therefore, the total response latency does not have a clear relationship with the number of RSUs in the simulation. To summarize, the simulation results indicate that the proposed approach provides a better solution for response latency optimization than the other two approaches across varying numbers of RSUs.

VII. Conclusion

Considerable focus has been dedicated to addressing task offloading and resource allocation challenges in Vehicular Edge Computing (VEC). However, certain issues remain inadequately tackled, as evidenced by prior research. In this paper, we propose a novel architecture known as the DT-VEC, where DTs are utilized to simulate the states of RSUs and the entire VEC system. In particular, a task offloading algorithm is put forward to minimize the total response latency for all tasks within the optimization period with the assistance of DT technologies. We also evaluate the algorithm in comparison with other greedy algorithms, and the simulation results reveal that it can achieve a better effect.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grant Number 62071327, 62271486 and 62071470. Huaming Wu is the corresponding author.

References

[1] C. Tang, W. Chen, C. Zhu, Q. Li, and H. Chen, "When cache meets vehicular edge computing: Architecture, key issues, and challenges," *IEEE Wirel. Commun.*, vol. 29, no. 4, pp. 56–62, 2022.

[2] X. Wang, L. T. Yang, L. Ren, Y. Wang, and M. J. Deen, "A tensor-based computing and optimization model for intelligent edge services," *IEEE Network*, vol. 36, no. 1, pp. 40–44, 2022.

[3] W. Sun, H. Zhang, R. Wang, and Y. Zhang, "Reducing offloading latency for digital twin edge networks in 6g," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12 240–12 251, 2020.

[4] X. Wang, L. Ren, R. Yuan, L. T. Yang, and M. J. Deen, "Qtt-dlstm: A cloud-edge-aided distributed lstm for cyber-physical-social big data," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2022.

[5] B. Fan, Y. Wu, Z. He, Y. Chen, T. Q. S. Quek, and C. Xu, "Digital twin empowered mobile edge computing for intelligent vehicular lane-changing," *IEEE Netw.*, vol. 35, no. 6, pp. 194–201, 2021.

[6] Y. Dai and Y. Zhang, "Adaptive digital twin for vehicular edge computing and networks," *Journal of Communications and Information Networks*, vol. 7, no. 1, pp. 48–59, 2022.

[7] C. Schwarz and Z. Wang, "The role of digital twins in connected and automated vehicles," *IEEE Intell. Transp. Syst. Mag.*, vol. 14, no. 6, pp. 41–51, 2022.

[8] H. Feng, D. Chen, and Z. Lv, "Blockchain in digital twins-based vehicle management in vanets," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 19 613–19 623, 2022.
[9] Z. Lv, J. Guo, A. K. Singh, and H. Lv, "Digital twins based VR simulation for accident prevention of intelligent vehicle," *IEEE Trans. Veh. Technol.*, vol. 71, no. 4, pp. 3414–3428, 2022.
[10] O. Chukhno, N. Chukhno, G. Araniti, C. Campolo, A. Iera, and A. Molinaro, "Placement of social digital twins at the edge for beyond 5g iot networks," *IEEE Internet Things J.*, vol. 9, no. 23, pp. 23 927–23 940, 2022.
[11] Z. Lv, D. Chen, H. Feng, H. Zhu, and H. Lv, "Digital twins in unmanned aerial vehicles for rapid medical resource delivery in epidemics," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 12, pp. 25 106–25 114, 2022.
[12] S. Wang, C. Tu, and J. Juang, "Automatic traffic modelling for creating digital twins to facilitate autonomous vehicle development," *Connect. Sci.*, vol. 34, no. 1, pp. 1018–1037, 2022.
[13] F. Sanfilippo, E. F. Langås, H. Z. Bukhari, and S. Robstad, "Pervasive and connected digital twins for edge computing enabled industrial applications," in *56th Hawaii International Conference on System Sciences, HICSS 2023, Maui, Hawaii, USA, January 3-6, 2023*, T. X. Bui, Ed. ScholarSpace, 2023, pp. 6789–6798.
[14] C. Tang, S. Xia, Q. Li, W. Chen, and W. Fang, "Resource pooling in vehicular fog computing," *J. Cloud Comput.*, vol. 10, no. 1, p. 19, 2021.
[15] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, 2016.
[16] C. Tang, C. Zhu, H. Wu, Q. Li, and J. J. P. C. Rodrigues, "Toward response time minimization considering energy consumption in caching-assisted vehicular edge computing," *IEEE Internet Things J.*, vol. 9, no. 7, pp. 5051–5064, 2022.