Dependency-Aware Task Scheduling in Vehicular Edge Computing

Yujiong Liu, Shangguang Wang, *Senior Member, IEEE,* Qinglin Zhao, Shiyu Du, Ao Zhou, *Member, IEEE,* Xiao Ma, *Member, IEEE,* and Fangchun Yang, *Senior Member, IEEE*

***Abstract*—Vehicular edge computing offers a new paradigm to improve vehicular services and augment the capabilities of vehicles. In this paper, we study the problem of task scheduling in vehicular edge computing, where multiple computation-intensive vehicular applications can be offloaded to road side units and each application can be further divided into multiple tasks with task dependency. The tasks can be scheduled to different mobile edge computing servers on road side units for execution to minimize the average completion time of multiple applications. Considering the completion time constraint of each application and the processing dependency of multiple tasks belonging to the same application, we formulate the multiple tasks scheduling problem as an optimization problem that is NP-hard. To solve the optimization problem, we develop an efficient task scheduling algorithm. The basic idea is to prioritize multiple applications and prioritize multiple tasks so as to guarantee the completion time constraints of applications and the processing dependency requirements of tasks. Numerical results demonstrate that our proposed algorithm can significantly reduce the average completion time of multiple applications compared with benchmark algorithms.**

***Index Terms*—Multiple applications, Task scheduling, Task dependency, Vehicular edge computing.**

I. Introduction

With the rapid advancement of internet of things and wireless communication technology, vehicles have been a significant component of mobile devices connecting to the Internet. Vehicles can run various computation-intensive applications, such as image-aided navigation, intelligent vehicle control, traffic management, in-vehicle entertainment and augmented vehicular reality. These computation-intensive applications not only require massive computation resources and storage resources to handle complicated data processing and storage operations, but also have stringent delay requirements. However, resource-constrained vehicles have not been ready to provide sufficient computation resources and storage resources to meet the resource requirements of these applications. This poses a significant challenge for vehicles to ensure their required quality of service [1], [2], [3], [4], [5].

In order to cope with the explosive computation resources and storage resources demands of vehicles, mobile cloud computing is envisioned as a promising paradigm [6]. Mobile cloud computing is a system that introduces cloud computing capability into mobile computing environments. In mobile cloud computing, vehicles can offload their computation-intensive applications to conventional resource-rich cloud servers via wireless networks, greatly extending their computation capabilities [7].

However, for computation-intensive and delay-sensitive applications, such as augmented vehicular reality, it is insufficient to offload the computation-intensive applications to conventional cloud servers, because conventional cloud servers are located far away from vehicles, the remote offloading will lead to unpredictable transmission delay and serious degradation of quality of service. To address the challenges, Mobile Edge Computing (MEC) is proposed as a promising solution that can provide cloud services at the edge of radio access networks. By providing services closer to vehicles, MEC can decrease the transmission delay of applications and alleviate massive computation resources requirements of vehicles greatly [8], [9], [10].

Furthermore, as an important use case of MEC, vehicular edge computing (VEC) has emerged as a new computing paradigm in the field of intelligent transportation system that has received significant attention in recent years, because it can extend cloud computation capability to the close proximity of vehicles. In VEC, to provide high level quality of service for vehicles, a large number of cloud services are deployed on Road Side Units (RSUs), as illustrated in Fig. 1, vehicles can access to RSUs via wireless networks. In this case, computation-intensive and delay-sensitive applications can be offloaded to MEC servers on RSUs for execution [11].

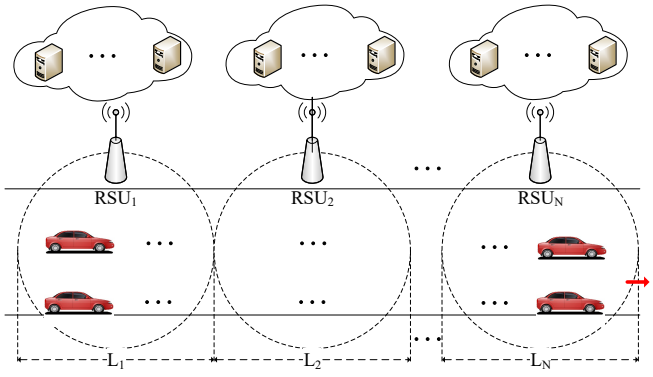


Fig. 1. The architecture of VEC.

There has been some research works on VEC. Zhang *et*

Yujiong Liu, Shangguang Wang, Shiyu Du, Ao Zhou, Xiao Ma, and Fangchun Yang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876 China.
E-mail: {yjliu, sgwang, dsy, aozhou, maxiao18, fcyan}@bupt.edu.cn.
Qinglin Zhao is with Faculty of Information Technology, Macau University of Science and Technology, Avenida WeiLong, Taipa, Macau, China.
E-mail: zqlict@hotmail.com.

al. [12] proposed a task offloading scheme based on Stackelberg game theory to maximize the utilities of both vehicles and MEC servers. Zhang *et al.* [13] introduced an efficient predictive combination-mode offloading mechanism to reduce offloading cost. Dai *et al.* [14] developed a joint optimal VEC server selection and offloading algorithm to maximize system utility. Zhou *et al.* [15] proposed an energy-efficient resource allocation algorithm based on alternating direction method of multipliers. Zhu *et al.* [16] developed a dynamic task allocation solution to ensure the quality of service. The prior studies assume that applications consisting of independent tasks are offloaded to RSUs for execution. However, task execution order depends on task dependency and the effect of task dependency on execution time of applications has not been considered in the previous research works. For an augmented vehicular reality system with the following major components: object tracking, model mapping, object recognition, perspective transformation and merging processing, there are some task dependencies among the components, e.g., only after one vehicle is tracked, the surrounding environmental model of the vehicle can be built and only after one vehicle is recognized, the process of perspective transformation and merging processing can be executed. To ensure that multiple vehicular applications can be completed in time, it is necessary to take task dependency into account for task scheduling policies design.

To overcome the above drawbacks, in this paper, we consider the task dependency and the completion time constraints when scheduling tasks into multiple MEC servers. The goal of the work is to identify task scheduling decision that minimizes the average completion time of multiple applications, subject to their respective completion time constraints. We first present a VEC architecture. Then, we specify the completion time constraint of each application and the task dependency requirements of tasks. Finally, we propose an efficient task scheduling algorithm to minimize the average completion time of multiple applications.

The main contributions of this work are as follows:

- We propose a VEC architecture which consists of multiple vehicles, multiple RSUs and multiple MEC servers. Each vehicle has a computation-intensive and delay-sensitive application. Each RSU is equipped with multiple MEC servers. Multiple vehicles can offload their computation-intensive and delay-sensitive applications to MEC servers on RSUs for execution where applications are independent of each other but tasks (belonging to the same application) have processing dependence.
- We formalize the task scheduling decision problem as an optimization problem which is NP-hard, and then propose an efficient Multiple Applications Multiple Tasks Scheduling (MAMTS) algorithm to solve the optimization problem. Furthermore, we prioritize multiple applications to meet their respective completion time constraints, and prioritize multiple tasks for satisfying their processing dependency requirements.
- We evaluate the proposed task scheduling algorithm with extensive simulations. The simulation results show that our proposed algorithm can significantly reduce the average completion time of multiple applications compared with benchmark algorithms.

The rest of the paper is organized as follows. We first briefly discuss related work in Section II. We then depict the system model in details in Section III. We next formulate the task scheduling decision problem as an optimization problem in Section IV. We further propose an efficient task scheduling algorithm to solve the optimization problem in Section V. We present the simulation results to evaluate the performance of the proposed algorithm in Section VI. Finally, we conclude the paper in Section VII.

II. Related Work

There are some research works on offloading computation-intensive tasks to nearby MEC servers. Guo *et al.* [17] considered the requirements of heterogeneous quality of service and principles of resource allocation. A hierarchical genetic algorithm and particle swarm optimization-based computation algorithm was proposed. You *et al.* [18] developed an energy-efficient resource-management policy, so as to minimize total mobile-energy consumption for the asynchronous MEC system. Ji *et al.* [19] studied the problem of energy-effective computation offloading and resource allocation, so as to maximize energy efficiency under the minimal quality of service constraint. Xu *et al.* [20] considered both dynamic service caching and task offloading. An efficient online and decentralized computation offloading algorithm was proposed. However, the above research works focus on how to make computation offloading decisions, none of them considers how to schedule applications on multiple MEC servers to improve the computation offloading performance.

The problem of task scheduling based on task dependency in distributed system was investigated in some recent works. Yao *et al.* [21] proposed a novel scheduling policy for Hadoop YARN systems, so as to improve the resource utilization and reduce the makespan of a given set of MapReduce jobs. Wu *et al.* [22] adopted genetic algorithm to solve the task scheduling problem. Topcuoglu *et al.* [23] developed two new static scheduling algorithms for heterogeneous systems. Geng *et al.* [24] focused on energy-efficient computation offloading for multi-core mobile devices. A heuristic algorithm was proposed to jointly solve computation offloading decisions and task scheduling problems. Sundar *et al.* [25] proposed to greedily schedule tasks to minimize application execution cost in a cloud computing system. Wang *et al.* [26] proposed a task scheduling approach for a mobile cloud computing system based on ant colony optimization algorithm. The research works assume that the servers have infinite capacity. However, this assumption is unrealistic, since MEC servers usually have limited resources. In our work, each RSU is equipped with multiple MEC servers with limited computation capacity.

Some studies focused on the problem of improving the performance of task scheduling in VEC. Sun *et al.* [27] considered instability of computation resource and heterogeneity of computation capability. A cooperative task scheduling scheme was proposed to improve computation efficiency. Zhu *et al.* [28] proposed an event-triggered dynamic task allocation

framework to meet the constraints on service latency, and quality loss. Qi *et al.* [29] designed a knowledge driven service offloading decision framework for internet of vehicle. The aforementioned schemes have not considered the influence of the priorities of different applications on the algorithm performance.

The aforementioned schemes cannot be directly applied to our proposed applications offloading scenario. Different from the aforementioned works, in this paper, we consider that multiple applications have their respective completion time constraints, there are task dependency requirements for multiple tasks belonging to the same application, on the other hand, MEC servers on RSUs have limited computation capacities. The tasks can be scheduled to different MEC servers with limited computation capacity for execution, to minimize the average completion time of multiple applications.

III. System Model

In this section, we present the system model, i.e., network model, application model, and computation model in details.

A. Network Model

In this section, we introduce the network model of the system. Fig. 1 shows the architecture of vehicular edge computing which consists of vehicles, RSUs and MEC servers. We consider that there are M vehicles arriving at the starting point of a unidirectional road and there are N RSUs along the unidirectional road. Each RSU is equipped with R MEC servers. The set of vehicles is denoted as $\mathcal{M} = \{1, 2, \cdots, M\}$, the set of RSUs is denoted as $\mathcal{N} = \{1, 2, \cdots, N\}$, the set of MEC servers on each RSU is denoted as $\mathcal{R} = \{1, 2, \cdots, R\}$. Based on the coverage range of each RSU, we can divide the road into corresponding N segments with different length $\{L_1, L_2, \cdots, L_N\}$, respectively. For ease of reference, we show the key notations used in this paper in Table I.

TABLE I. Key Notations

Notation	Description
$\mathcal{M}, M$	set / number of vehicles
$\mathcal{N}, N$	set / number of RSUs
$\mathcal{R}, R$	set / number of MEC servers on each RSU
m	the vehicle index $m \in \mathcal{M}$
n	the RSU index $n \in \mathcal{N}$
r	the MEC server index $r \in \mathcal{R}$
T_m	the mth application
$T_{m,i}$	the ith task of application T_m
$\mathcal{I}, I$	set / number of tasks of application T_m
i	the task index $i \in \mathcal{I}$
$x_{m,i,r}$	the scheduling decision variable of task $T_{m,i}$
$RT_{m,i}$	the ready time of task $T_{m,i}$
$AFT_{m,i}$	the actual completion time of task $T_{m,i}$
$EST_{m,i,r}$	the earliest start time of task $T_{m,i}$
$EFT_{m,i,r}$	the earliest finish time of task $T_{m,i}$

We consider that vehicles are moving at a constant speed v. The vehicles moving within the nth road segment can access the nth RSU via wireless channel, $n \in \mathcal{N}$ represents one RSU. We assume that each vehicle has a computation-intensive and delay-sensitive application to be executed within a stringent completion time constraint. Each vehicle can offload its application to one RSU for execution. Each application can be described in three terms as $T_m = \{d_m, b_m, t_m^{max}\}, m \in \mathcal{M}$ represents one vehicle, where d_m is the size of input data describing some information of application T_m, b_m is the amount of computation resource required to complete application T_m, and t_m^{max} is the maximum delay allowed to complete application T_m.

B. Application Model

In this section, we introduce the application model of the system. We assume that each application can be partitioned into multiple tasks. The tasks can be mutually dependent, i.e., the tasks can not be scheduled simultaneously and some tasks may need input from other tasks. We refer to a task as the unit of computation. Each task can be scheduled to any MEC server for execution. Each application T_m can be modeled as a directed acyclic graph $\mathcal{G} =< \mathcal{I}, \mathcal{E} >$, where $\mathcal{I}$ is the set of task nodes and $\mathcal{E}$ is the set of edges. Let $I = |\mathcal{I}|$ denote the total number of tasks belonging to application T_m. In the task graph, node $T_{m,i}$ represents the ith task belonging to application T_m, edge $(T_{m,i}, T_{m,j})$ represents the task dependency that task $T_{m,j}$ can not be scheduled until task $T_{m,i}$ has been completed, $i, j \in \mathcal{I},$. In one given task graph, the task without any immediate predecessor node is known as entry task, and the task without any immediate successor node is known as exit task. Note that tasks belonging to different applications are independent of each other, however, tasks belonging to the same application may have task dependency requirements.

Fig. 2 shows an example of task graphs of two applications. In Fig. 2, task $T_{1,1}$ and task $T_{2,1}$ represent the 1th task of application T_1 and application T_2, respectively. Task $T_{1,1}$ and task $T_{2,1}$ are entry tasks of application T_1 and application T_2, respectively. Task $T_{1,10}$ and task $T_{2,8}$ are exit tasks of application T_1 and application T_2, respectively. Task $T_{1,2}$ is immediate predecessor task of task $T_{1,5}$ and task $T_{1,6}$. Task $T_{1,5}$ and task $T_{1,6}$ are immediate successor tasks of task $T_{1,2}$. Task $T_{1,5}$ and task $T_{1,6}$ are started until task $T_{1,2}$ has been completed. Task $T_{2,2}$ and task $T_{2,3}$ are immediate predecessor tasks of task $T_{2,5}$. Task $T_{2,5}$ is immediate successor task of task $T_{2,2}$ and task $T_{2,3}$. Task $T_{2,5}$ is started until task $T_{2,2}$ and task $T_{2,3}$ have been completed.

C. Computation Model

In this section, we introduce the computation model of the system. To calculate the completion time of one application, we need to first model the execution process of every application. Specifically, the completion time of each application consists of four parts, i.e., vehicle movement time, data transmission time, application computing time and result transmission time. Vehicle movement time is the time taken for one vehicle from its starting point to the coverage range of one RSU. Data transmission time is the time that input data of one application is transmitted from one vehicle to one RSU.

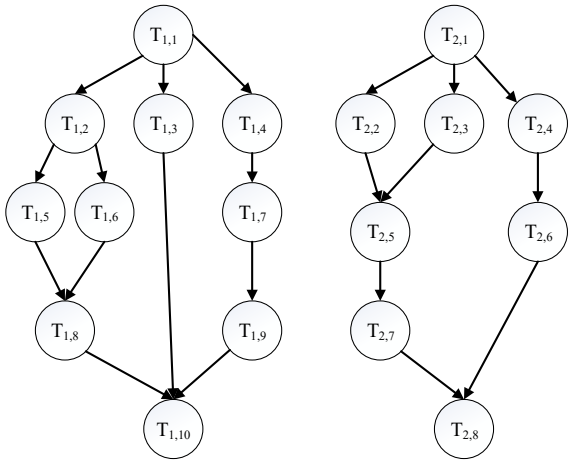


Fig. 2. An example of task graphs of two applications.

Application computing time is the time that one application is completed on one RSU. Result transmission time is the time that the computation result of one application is transmitted from one RSU to one vehicle. Since the size of computation result of one application is often much smaller than its input data size, therefore, the computation result transmission time can be ignored. When application T_m is offloaded from vehicle m to RSU n, we denote $t_m^{process}$ as the total completion time that application T_m is processed on RSU n. Next, we introduce the models of vehicle movement time, data transmission time and application computing time.

1) Movement Time: Due to the high mobility of vehicles, when vehicle m moves from its starting point to the coverage area of RSU n, application T_m can be offloaded to RSU n from vehicle m. We assume that vehicle m has passed $n-1$ road segments from its initial location in the vehicle heading direction. We denote t_m^{mov} as the movement time of vehicle m, i.e., the time taken for vehicle m from its starting point to the coverage range of RSU n, thus, vehicle movement time t_m^{mov} can be given as

$$t_m^{mov} = \sum_{k=1}^{n-1} \frac{L_k}{v} \tag{1}$$

2) Transmission Time: We consider that the wireless communication between vehicles and RSUs is based on Orthogonal Frequency Division Multiple Access. We denote t_m^{send} as the transmission time of input data d_m of application T_m, denote p_m as the transmission power of vehicle m, denote g_m^n as the channel gain between vehicle m and RSU n, denote s_m^n as the data transmission rate of the link between vehicle m to RSU n, thus, data transmission rate s_m^n can be given as

$$s_m^n = Wlog(1 + \frac{p_m g_m^n}{N_0}) \tag{2}$$

where W represents the bandwidth of the link between vehicle m to RSU n, N_0 represents the noise power. Thus, data transmission time t_m^{send} required by vehicle m for uploading its application T_m with size d_m can be given as

$$t_m^{send} = \frac{d_m}{s_m^n} \tag{3}$$

3) Computing Time: We consider that computation resource of each RSU is limited. Since multiple vehicles may select the same RSU as their offloading target, thus, we need to allocate computation resources for multiple applications to meet their respective completion time constraints.

We assume that each MEC server on RSUs can only execute one task at a time, therefore, other tasks assigned to the same MEC server may not be processed immediately, due to the existing task being processed in the same MEC server. We denote $AFT_{m,i}$ as the actual time that task $T_{m,i}$ is completed on MEC servers, where $T_{m,i}$ represents the ith task of application T_m. We can consider this problem based on the project management theory [30]. Since there may be task dependency requirements among the tasks belonging to the same application, all immediate predecessor tasks of task $T_{m,i}$ must have been completed before task $T_{m,i}$ is started. Next, we definite the ready time of task $T_{m,i}$.

The ready time of one task is the earliest time that all its immediate predecessor tasks have been completed, thus, the ready time $RT_{m,i}$ of task $T_{m,i}$ can be given as

$$RT_{m,i} = \max_{T_{m,h} \in pre(T_{m,i})} AFT_{m,h} \tag{4}$$

where $pre(T_{m,i})$ represents the set of immediate predecessor tasks of task $T_{m,i}$, $T_{m,h}$ must have been completed before task $T_{m,i}$ is started.

Furthermore, we assume that when MEC server r is idle,$r \in \mathcal{R}$ represents one MEC server on one RSU, task $T_{m,i}$ can be scheduled to MEC server r. If another task has been executing on MEC server r, task $T_{m,i}$ must wait in a queue until MEC server r is available. We denote $AT_{m,i,r}$ as the earliest time that MEC server r is available for task $T_{m,i}$. When one task has been ready and one MEC server is available for the task, the task can be started on the MEC server.

The earliest start time of one task is the earliest time that one task has been started after the task has been ready and one MEC server is available for the task, thus, the earliest start time $EST_{m,i,r}$ of task $T_{m,i}$ on MEC server r can be given as

$$EST_{m,i,r} = \max\{RT_{m,i}, AT_{m,i,r}\} \tag{5}$$

We denote $ET_{m,i,r}$ as the time that task $T_{m,i}$ is executed on MEC server r, execution time $ET_{m,i,r}$ can be given as

$$ET_{m,i,r} = \frac{b_{m,i}}{f_r} \tag{6}$$

where $b_{m,i}$ represents the amount of computation resource required to complete task $T_{m,i}$, f_r represents the computation capability of MEC server r.

The earliest finish time of one task is the earliest time that one task has finished its execution on one MEC server, thus, the earliest finish time $EFT_{m,i,r}$ of task $T_{m,i}$ on MEC server r can be given as

$$EFT_{m,i,r} = EST_{m,i,r} + ET_{m,i,r} \tag{7}$$

After task $T_{m,i}$ is scheduled to MEC server r, earliest finish time $EFT_{m,i,r}$ is equal to its actual finish time $AFT_{m,i}$.

After all tasks of application T_m are scheduled to MEC

servers, finish time of exit task is equal to the actual computing time of application T_m. We denote $T_{m,I}$ as the exit task of application T_m, thus, the computing time t_m^{comp} of application T_m on RSU n can be given by

$$t_m^{comp} = AFT_{m,I} \tag{8}$$

According to Eq.(1), (3), and (8), completion time $t_m^{process}$ can be written as follows:

$$t_m^{process} = t_m^{mov} + t_m^{send} + t_m^{comp} \tag{9}$$

IV. Problem Formulation

In this section, we formalize the task scheduling decision problem as an optimization problem. To ensure that all the tasks can be executed on MEC servers, we first define a scheduling decision variable $x_{m,i,r}$ as follows:

$$x_{m,i,r} = \begin{cases} 1 & \text{if task } T_{m,i} \text{ can be executed on MEC server } r, \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

where $x_{m,i,r} = 0$ represents that task $T_{m,i}$ can not be executed on MEC server r. Moreover, to ensure that each task is scheduled to only one MEC server, according to Eq.(10), we can define such a constraint $\sum_{r=1}^{R} x_{m,i,r} = 1$.

For task scheduling, we define a binary variable $y_{h,i}$ to specify the task scheduling order as follows:

$$y_{h,i} = \begin{cases} 1 & \text{if task } h \text{ is scheduled before task } i, \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

where $y_{h,i} = 1$ represents that task i is not scheduled until task h has been scheduled.

Based on the above definition variables, the earliest start time $EST_{m,i,r}$ of task $T_{m,i}$ on MEC server r should meet the following constraint:

$$EST_{m,i,r} \geq x_{m,i,r} \cdot x_{s,h,r} \cdot y_{h,i} \cdot EFT_{s,h} \tag{12}$$

That is, if task $T_{m,i}$ is scheduled to MEC server r, all other tasks that are scheduled to the same MEC server r before task $T_{m,i}$ should have been completed.

Besides, the task dependency among the tasks belonging to the same application should meet the following constraint:

$$AFT_{m,i} \geq y_{h,i} \cdot AFT_{m,h} \tag{13}$$

That is, if task $T_{m,h}$ and task $T_{m,i}$ belong to the same application T_m and task $T_{m,h}$ is immediate predecessor task of task $T_{m,i}$, task $T_{m,i}$ is not completed until task $T_{m,h}$ has been completed.

Furthermore, the finish time of exit task of application T_m should meet the following constraint:

$$AFT_{m,I} \leq t_m^{max} - t_m^{mov} - t_m^{send} \tag{14}$$

ensures that application T_m can be finished within its completion time constraint t_m^{max}.

The objective is to identify the task scheduling decision that minimizes the average completion time of multiple applications, subject to their respective completion time constraints. Thus, the task scheduling decision problem can be formulated as an optimization problem as follows:

$$min \frac{1}{m}[\quad \sum_{m=1}^{M} (t_m^{mov} + t_m^{send}) + \sum_{m=1}^{M} \sum_{i=1}^{I} \sum_{r=1}^{R} x_{m,i,r} \cdot EFT_{m,i,r}]$$
$$subject\ to\ Eq.(10), (11), (12), (13), and\ (14) \tag{15}$$

The problem in Eq.(15) is a mixed-integer nonlinear programming problem. It is known from [31] that a mixed-integer nonlinear programming problem is NP-hard, therefore, the optimization problem in Eq.(15) is also NP-hard. To solve the problem with low complexity, we propose an efficient task scheduling algorithm in next section.

V. Proposed Algorithm and Analysis

In this section, we first propose an efficient task scheduling algorithm, and then analyze its computational complexity.

A. Proposed Algorithm

In this section, we present the details of our proposed algorithm, which has three phases. First, we construct an application priority queue for multiple applications without violating their respective completion time constraints. Second, we construct a task priority queue for multiple tasks without violating their respective completion time constraints. Last, we schedule tasks to multiple MEC servers to reduce the average completion time of multiple applications.

1) Application Prioritization: To ensure that all applications are finished within their respective completion time deadlines, we first construct an application priority queue. An application priority queue is a queue that keeps multiple applications in a decreasing order of application priority. Intuitively, applications with more stringent completion time deadlines should have higher priority to be scheduled, i.e., one application with the minimal completion time deadline should be given the highest priority and reside on the head of an application priority queue. An application priority queue can be obtained by increasing order of completion time deadlines of multiple applications.

2) Task Prioritization: Since there are task dependency requirements among the tasks belonging to the same application, thus, we identify the completion time constraints of all tasks. This can be achieved by computing latest finish time of all tasks. The latest finish time of one task is treated as its individual completion time constraint. Thus, we construct a task priority queue. A task priority queue is a queue that keeps the tasks in a decreasing order of task priority. Intuitively, tasks with more stringent completion time constraints should have higher priority to be scheduled, i.e., one task with the minimal completion time constraint should be given the highest priority and reside on the head of a task priority queue. Next, we give some variables to obtain a task priority queue.

The latest finish time of one task is the latest time that one task has finished its execution on MEC servers, thus, the latest finish time $LFT_{m,i}$ of task $T_{m,i}$ can be given as

$$LFT_{m,i} = \min_{T_{m,j} \in suc(T_{m,i})} (LFT_{m,j} - ET_{m,j}^{min}) \tag{16}$$

where $suc(T_{m,i})$ is the set of all immediate successor tasks of task $T_{m,i}$, $ET_{m,j}^{min}$ is the minimal time that task $T_{m,j}$ is executed on MEC servers, i.e., $ET_{m,j}^{min} = \min_{1 \le r \le R} ET_{m,j,r}$.

The completion time constraint of application T_m is t_m^{max}, thus, the latest finish time of exit task of application T_m is also t_m^{max}. For one task, its latest finish time is its tight deadline. If one task has not been finished before its latest finish time, it may not meet the completion time constraint of the application.

The latest start time of one task is the latest time that one task has been started on MEC servers, thus, the latest start time $LST_{m,i}$ of task $T_{m,i}$ can be derived from its latest finish time $LFT_{m,i}$ as follows

$$LST_{m,i} = LFT_{m,i} - ET_{m,i}^{min} \tag{17}$$

The latest start time of one task can be used to measure its emergency. One task with more stringent latest start time should be started earlier. If one task is exactly finished at its latest finish time, other tasks may not be finished within their respective deadlines. Thus, we denote $SLFT$ as the slack latest finish time that one task has slack time to complete its execution on MEC servers. $SLFT_{m,i}$ of task $T_{m,i}$ can be given as

$$SLFT_{m,i} = \min_{T_{m,j} \in suc(T_{m,i})} (LFT_{m,j} - ET_{m,j}^{max}) \tag{18}$$

where $ET_{m,j}^{max}$ is the maximum time that task $T_{m,j}$ is executed on MEC servers, i.e., $ET_{m,j}^{max} = \max_{1 \le r \le R} ET_{m,j,r}$. Similarly, the slack latest finish time of exit task of application T_m is also t_m^{max}.

For one task, if it is finished within its latest finish time, this may cause the application to fail to be finished within the application deadline. Because the slack latest finish time of one task is usually earlier than its latest finish time, if all the immediate predecessor tasks of one task have been finished before their respective slack latest finish time, the task can always be finished before its latest finish time. Furthermore, this can ensure that the application can be finished within its delay constraint. Therefore, for one task, we can compute its latest start time by replacing its latest finish time with its slack latest finish time.

One more urgent task should have higher priority to be scheduled. Therefore, for one task, its latest start time can be used to measure its priority, i.e., one task with more stringent latest start time should have higher priority to be scheduled. Thus, we can obtain a task priority queue by sorting latest start time of tasks.

3) Task Scheduling: The basic idea is to choose one task that has the highest task priority and schedule it to one MEC server that can minimize its completion time. The input of task scheduling algorithm is multiple applications. We first create an application priority queue by increasing order of multiple applications. Moreover, we create a task priority queue by increasing order of multiple tasks.

When multiple tasks are scheduled, if one task with lower task priority belonging to one application has been ready earlier than one task with higher task priority belonging to other applications, we should first schedule the task that has been ready rather than strictly following the task priority order. Thus, we can have a task waiting list to save the tasks that have been not ready but have higher task priority than current task.

The detailed steps of task scheduling are as follows:

- Choose task $T_{m,i}$ with highest task priority from the task priority queue obtained by task priority sorting operation. Compute the ready time $RT_{m,i}$ of task $T_{m,i}$ according to Eq.(4). If task $T_{m,i}$ has been ready, we compute its execution time $ET_{m,i}$ on all MEC servers according to Eq.(6). If task $T_{m,i}$ has been not ready, we save it in the task waiting list.
- Schedule task $T_{m,i}$ to obtain its minimal execution time on MEC servers, however, because any task in the task waiting list may not be finished within its deadline, we may obtain a non feasible task scheduling scheme. When task $T_{m,i}$ is scheduled to multiple MEC servers, we compute the execution time of other tasks in the task waiting list on MEC servers according to Eq.(7) supposing task $T_{m,i}$ is scheduled. For task $T_{m,j}$, if it can not be completed within its $SLFT_{m,j}$, task $T_{m,j}$ should be scheduled to MEC servers before task $T_{m,i}$.
- For task $T_{m,i}$, if it has been scheduled to MEC servers, we will update the task scheduling sequence and delete it from the task priority queue, and return to the first step until all tasks have been scheduled to MEC servers.

The detailed description of the proposed algorithm is given in Algorithm 1. As an example, we perform the task scheduling algorithm on the task graphs given in Fig. 2. In this example, we set the completion time deadlines of two applications $t_1^{max} = 1$, $t_2^{max} = 2$, respectively. Based on respective completion time deadlines of the two applications, we can construct an application priority queue (T_1, T_2). Based on the task dependency of multiple tasks, we can construct an initial task priority queue $(T_{1,1}, T_{1,2}, T_{1,3}, T_{1,4}, T_{1,5}, T_{1,6}, T_{1,7}, T_{1,8}, T_{1,9}, T_{1,10}, T_{2,1}, T_{2,2}, T_{2,3}, T_{2,4}, T_{2,5}, T_{2,6}, T_{2,7}, T_{2,8})$. Based on the tasks' LST, we can construct a new task priority queue $(T_{1,1}, T_{2,1}, T_{1,2}, T_{1,4}, T_{1,3}, T_{2,2}, T_{2,3}, T_{2,4}, T_{1,7}, T_{2,6}, T_{1,6}, T_{1,5}, T_{2,5}, T_{1,9}, T_{1,8}, T_{2,7}, T_{1,10}, T_{2,8})$. For example, after task $T_{1,2}$ has been scheduled to one MEC server, task $T_{1,5}$ and task $T_{1,6}$ should wait until $T_{1,2}$ has been finished. Assuming that task $T_{1,7}$ has been ready, task $T_{1,7}$ should be scheduled to one MEC server before task $T_{1,5}$ and task $T_{1,6}$. Since the objective is to minimize the average completion time of two applications, when any MEC server is available, tasks should be scheduled to idle MEC servers to meet respective completion time deadlines of two applications.

B. Complexity Analysis

In this section, we analyze the computational complexity of our proposed algorithm. The computational complexity of Algorithm 1 mainly contains three parts, i.e., generating an application priority queue stage, generating a task priority queue stage and task scheduling stage. The complexity of generating an application priority queue stage is determined by line 2, it has the complexity of $O(M)$ to compute the

Algorithm 1 MAMTS Algorithm

Input: Vehicular applications, task graphs
Output: Task scheduling sequence TSS

1: **Initialize** task priority queue $TPQ = \Phi$, $TSS = \Phi$, waiting list $list = \Phi$;
2: **Sort** applications in increasing order by completion time deadlines of applications;
3: **Sort** tasks in increasing order by latest start time of tasks;
4: **function** $MainSchedule$
5: **while** $TPQ \neq \Phi$ **do**
6: Choose task $T_{m,i} = TPQ.head$;
7: Calculate $RT_{m,i}$ from Eq.(4);
8: **if** task $T_{m,i}$ has been not ready **then**
9: Save task $T_{m,i}$ in $list$;
10: **else**
11: $ScheduleTask\ T_{m,i}$;
12: **end if**
13: Update $T_{m,i}$;
14: Update TPQ;
15: **end while**
16: **end function**
17: **function** $ScheduleTask\ T_{m,i}$
18: $GetTaskFinishTime\ T_{m,i}$;
19: **for all** r **do**
20: Find the index of MEC server r such that $\min_{1 \leq r \leq R} \quad ET_{m,i,r}$;
21: Set the index of MEC server $Server_r$;
22: **end for**
23: **if** $list = \Phi$ **then**
24: $ScheduleTaskToServer(T_{m,i}, Server_r)$;
25: **end if**
26: **for** task $T_{n,j}$ in $list$ **do**
27: $GetTaskFinishTime\ T_{n,j}$;
28: **for all** r **do**
29: **if** $AFT_{n,j,r} > SLFT_{n,j}$ **then**
30: $ScheduleTask\ T_{n,j}$;
31: **end if**
32: **end for**
33: **end for**
34: $ScheduleTaskToServer(T_{m,i}, Server_r)$;
35: **end function**
36: **function** $GetTaskFinishTime\ T_{m,i}$
37: **for all** r **do**
38: **if** MEC server $Server_r$ is idle **then**
39: Calculate $EFT_{m,i,r}$ from Eq.(7);
40: **end if**
41: **end for**
42: **return** $EFT_{m,i,r}$;
43: **end function**
44: **function** $ScheduleTaskToServer(T_{m,i}, Server_r)$
45: Schedule task $T_{m,i}$ to MEC server $Server_r$;
46: Update TSS;
47: Delete $T_{m,i}$ from TPQ;
48: **end function**

ranking metrics of all applications, $O(M \log M)$ to sort the applications. The complexity of generating a task priority queue stage is determined by line 3, it has the complexity of $O(Z)$ to sort the tasks, $O(Z \log Z)$ to sort the tasks, where $O(Z)$, $Z = MI$ represents the total number of tasks belonging to M applications. In the task scheduling stage, the dominating operation part is the while loop in lines 5-15. The complexity of the while loop is mainly determined by line 5 and line 11. The complexity of line 5 is $O(Z \log Z)$, the complexity of line 11 is determined by lines 18-34, the complexity of lines 18-34 is $O(ZR)$. Thus, the complexity of the task scheduling stage is $O(Z^2R)$. Therefore, the complexity of Algorithm 1 is $O(M \log M) + O(Z \log Z) + O(Z^2R) = O(Z^2R)$.

VI. Performance Evaluations

In this section, we provide simulation results to verify the performance of our proposed algorithm. In the following, we first introduce the simulation setup, and then present the comparison on average completion time of multiple applications with benchmark algorithms, further present the comparison on completion rate of applications with benchmark algorithms, finally, we analyze the impact of various parameters on the performance of the proposed algorithm.

A. Simulation Setup

In this section, based on the existing research works [14], [32], [33], we set various parameters for the simulations. We consider a scenario that 5 RSUs are uniformly located along a 100-meter unidirectional road. Each RSU is equipped with 5 MEC servers. Assume that there are 20 arriving vehicles on the unidirectional road, and the vehicles are running at a constant speed 120 km/hr. The input data sizes of applications are randomly drawn from the interval $[100, 300]$ KB. The completion time deadlines of applications are randomly drawn from the interval $[8, 10]$ seconds. For multiple applications, task graphs are randomly generated. The transmission power of each vehicle is 100 mW. The channel bandwidth between vehicles and RSUs is 5 MHz. The channel gain between vehicles and RSUs is 20^{-4}. The noise power of the system is 10^{-10} mW. The computation resources of MEC servers are randomly assigned from the interval $[1, 5]$ GHz. The computation resource requirements of the tasks are randomly assigned from the interval $[0.1, 0.4]$ GHz.

To verify the performance of our proposed algorithm, we conduct simulations based on randomly generated task trees in terms of directed acyclic graph structure of multiple applications [34]. Furthermore, we introduce the following benchmark algorithms.

- ***Greedy Scheduling (GS)***: Select tasks from top to bottom along the acyclic graph structure of multiple applications and schedule each task to one MEC server where it has the least execution time.
- ***Random Scheduling (RS)***: Select tasks from top to bottom along the acyclic graph structure of multiple applications and schedule each task to a randomly selected MEC server.

- ***Ant Colony Scheduling (ACS)***: Select tasks from top to bottom along the acyclic graph structure of multiple applications and schedule the tasks to MEC servers based on ant colony optimization algorithm [26].

B. Comparison of Average Completion Time

In this simulation, we compare our proposed algorithm with the aforementioned benchmark algorithms, with respect to the average completion time of multiple applications under their deadlines.

Fig. 3 shows the comparison on the average completion time of multiple applications versus their different deadlines. As shown in Fig. 3, we can observe that the average completion time of multiple applications is reduced by 36%, 26% and 2% comparing with GS, RS and ACS, respectively. This is because our proposed algorithm considers the priorities of both multiple applications and multiple tasks, therefore, computation resources can be fully utilized, however, benchmark algorithms have not considered the impact of priorities of both applications and tasks on completion time of applications and can not make the best of computation resources.

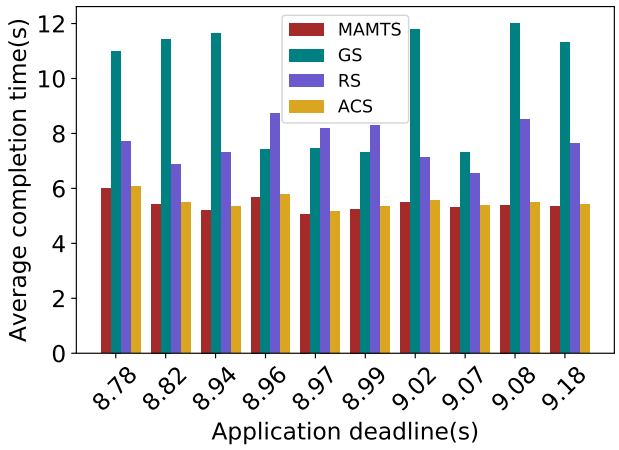


Fig. 3. Comparison of the average completion time under different application deadlines. The average completion time represents the average completion time of multiple applications, application deadline represents the average deadline of multiple applications. With our proposed algorithm, the average completion time of multiple applications is significantly reduced comparing with benchmark algorithms.

C. Comparison of Completion Rate

In this simulation, we compare our proposed algorithm with the benchmark algorithms, with respect to the completion rate, the percentage of schedules ensuring that all applications are completed under their deadlines.

Fig. 4 shows the comparison on the completion rate of multiple applications versus their different deadlines. As shown in Fig. 4, we can observe that with our proposed algorithm, the completion rate of multiple applications is higher than benchmark algorithms, the completion rate reaches 100%. This is because our proposed algorithm considers the priorities of both multiple applications and multiple tasks, therefore, high priority tasks are prioritized, it is easier to find a schedule to meet all application deadlines, however, benchmark algorithms have not considered the impact of priorities of both applications and tasks on completion time of applications.

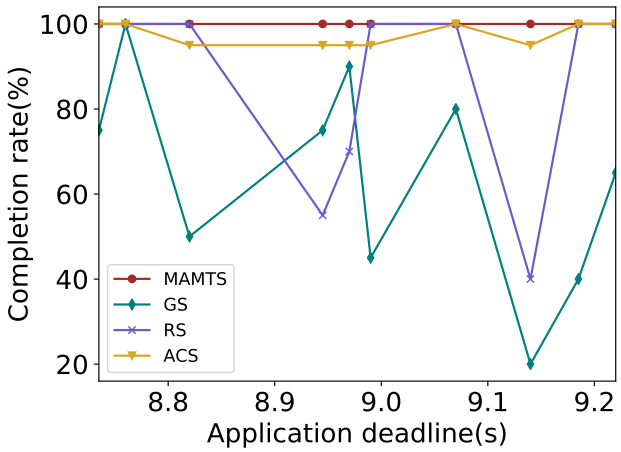


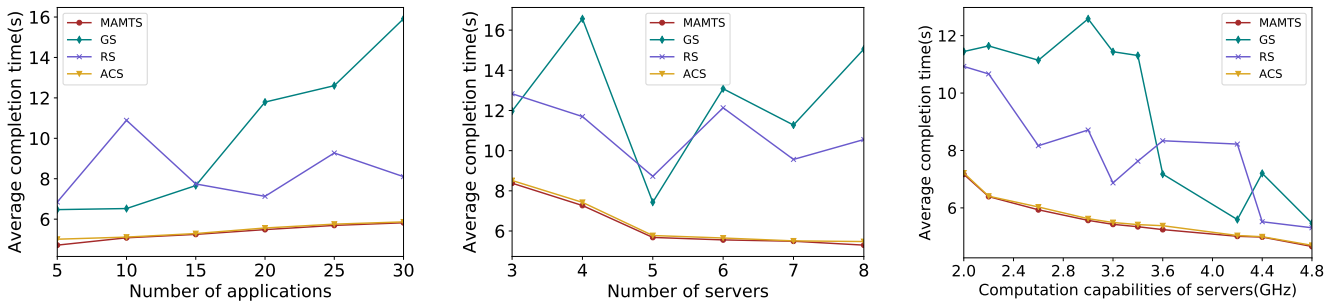
Fig. 4. Comparison of completion rate under different application deadlines. The completion rate represents the percentage of schedules that all applications are completed under their deadlines, application deadline represents the average deadline of multiple applications. With our proposed algorithm, the completion rate is much higher than benchmark algorithms.

D. Studies on the Parameters

In this section, we study the impact of various parameters on the performance of our proposed algorithm, i.e., the number of applications, the number of MEC servers on each RSU and computation capabilities of MEC servers.

1) Impact of the Number of Applications: Fig. 5(a) shows the impact of the number of applications M on the average completion time of multiple applications. To show its impact clearly, we vary the the number of applications M from 5 to 30 with a step value of 5. From Fig. 5(a), we can observe that the average completion time of multiple applications slowly increases as the number of applications M increases using our proposed algorithm. This is due to the fact that multiple applications compete for limited computation resources to meet their respective completion time constraints. This observation indicates that the average completion time of multiple applications is substantially influenced by the number of applications M under limited computation resources. The average completion time of multiple applications changes dramatically as the number of applications increases using benchmark algorithms.

2) Impact of the Number of MEC Servers: Fig. 5(b) shows the impact of the number of MEC servers R on the average completion time of multiple applications. To show its impact, we vary the the number of MEC servers R from 3 to 8 with a step value of 1. From Fig. 5(b), we can see that the average completion time of multiple applications significantly decreases with the increasing of the number of MEC servers R using our proposed algorithm. From Fig. 5(b), we can observe that when the number of MEC servers R increases from 3 to 5, the average completion time of multiple applications is significantly reduced and when the number of MEC servers R



(a) Impact of the number of applications M (b) Impact of the number of servers R (c) Impact of computation capabilities of servers f

Fig. 5. Impact of the parameters. (a)With our proposed algorithm, the average completion time of multiple applications slowly increases with the number of applications M. (b)With our proposed algorithm, the average completion time of multiple applications significantly decreases with the increase of the number of MEC servers R on each RSU. (c)With our proposed algorithm, the average completion time of multiple applications significantly decreases with the increase of computation capabilities of MEC servers f on each RSU.

increases from 5 to 8, the average completion time of multiple applications slowly decreases, that is because when the number of MEC servers R is 5, computation resources are sufficient to process the multiple applications. The average completion time of multiple applications changes sharply as the number of MEC servers increases using benchmark algorithms.

3) Impact of Computation Capability of MEC Servers: Fig. 5(c) shows the impact of computation capabilities of MEC servers f on the average completion time of multiple applications. We set that computation capabilities of MEC servers f are randomly assigned from the interval $[1, 5]$ GHz. It can be observed that the average completion time of multiple applications significantly decreases when computation capabilities of MEC servers f increase using our proposed algorithm, that is because when computation capabilities of MEC servers f increase, there are more computation resources to process multiple applications, thus, the average completion time of multiple applications can be significantly reduced. The average completion time of multiple applications changes sharply as computation capabilities of MEC servers increase using benchmark algorithms.

VII. Conclusion and Future Work

In this paper, we have studied a task scheduling problem for a VEC system which consists of multiple vehicles, multiple RSUs, and multiple MEC servers to reduce the average completion time of multiple applications. Each application has its respective completion time constraint. There are task dependency requirements among multiple tasks belonging to the same application. We have formulated the task scheduling decision problem as an optimization problem. To solve the optimization problem, we propose a MAMTS algorithm. Our proposed algorithm can sort multiple applications and multiple tasks to obtain an effective solution. Simulation results demonstrate that our proposed algorithm, outperforming the benchmark algorithms, can significantly reduce the average completion time of multiple applications while satisfying their respective completion time constraints.

We discuss the limitations of our research work which may inspire the future research work. For one thing, our research work does not consider the task communication delay, as we believe that the task processing result is small and the task transmission time is short. In reality, there may be data communication between tasks on different MEC servers. An optimal solution is to take into account the task communication delay. For another, our proposed algorithm lacks the verification of a large number of historical data. We plan to collect a large amount of historical data, and verify the performance of our proposed algorithm using real historical data. Solving our research problem based on some artificial intelligence algorithms is part of future work.

References

[1] E. Lee, E. Lee, M. Gerla, and S. Y. Oh, "Vehicular cloud networking: architecture and design principles," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 148–155, Feb. 2014.

[2] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[3] W. Quan, N. Cheng, M. Qin, H. Zhang, H. A. Chan, and X. Shen, "Adaptive transmission control for software defined vehicular networks," *IEEE Wireless Communications Letters*, vol. 8, no. 3, pp. 653–656, Jun. 2019.

[4] G. Liu, W. Quan, N. Cheng, H. Zhang, and S. Yu, "Efficient DDoS attacks mitigation for stateful forwarding in Internet of Things," *Journal of Network and Computer Applications*, vol. 130, pp. 1–13, Mar. 2019.

[5] W. Quan, Y. Liu, H. Zhang, and S. Yu, "Enhancing crowd collaborations for software defined vehicular networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 80–86, Aug. 2017.

[6] M. Chen, B. Liang, and M. Dong, "Multi-user multi-task offloading and resource allocation in mobile cloud systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 10, pp. 6790–6805, Oct. 2018.

[7] A. U. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 393–413, First Quarter 2014.

[8] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[9] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, Third Quarter 2017.

[10] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.

[11] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.

[12] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang, "Optimal delay constrained offloading for vehicular edge computing networks," in *Proc. of IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[13] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, Jun. 2017.

[14] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377–4387, Jun. 2019.

[15] Z. Zhou, P. Liu, Z. Chang, C. Xu, and Y. Zhang, "Energy-efficient workload offloading and power control in vehicular edge computing," in *Proc. of IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, Apr. 2018, pp. 191–196.

[16] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Yla-Jaaski, "Fog following me: Latency and quality balanced task allocation in vehicular fog computing," in *Proc. of 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, Jun. 2018, pp. 1–9.

[17] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2651–2664, Dec. 2018.

[18] C. You, Y. Zeng, R. Zhang, and K. Huang, "Asynchronous mobile-edge computation offloading: Energy-efficient resource management," *IEEE Transactions on Wireless Communications*, vol. 17, no. 11, pp. 7590–7605, Nov. 2018.

[19] L. Ji and S. Guo, "Energy-efficient cooperative resource allocation in wireless powered mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4744–4754, Jun. 2019.

[20] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2018, pp. 207–215.

[21] Y. Yao, J. Wang, B. Sheng, J. Lin, and N. Mi, "HaSTE: Hadoop YARN scheduling based on task-dependency and resource-demand," in *Proc. of IEEE 7th International Conference on Cloud Computing*, Jun. 2014, pp. 184–191.

[22] A. S. Wu, H. Yu, S. Jin, K. Lin, and G. Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 824–834, Sep. 2004.

[23] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[24] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation offloading for multicore-based mobile devices," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2018, pp. 46–54.

[25] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2018, pp. 37–45.

[26] T. Wang, X. Wei, C. Tang, and J. Fan, "Efficient multi-tasks scheduling algorithm in mobile cloud computing with time constraints," *Peer-to-Peer Networking and Applications*, vol. 11, no. 4, pp. 793–807, Jul. 2018.

[27] F. Sun, F. Hou, N. Cheng, M. Wang, H. Zhou, L. Gui, and X. Shen, "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11 049–11 061, Nov. 2018.

[28] C. Zhu, J. Tao, G. Pastor, Y. Xiao, Y. Ji, Q. Zhou, Y. Li, and A. Yl-Jski, "Folo: Latency and quality optimized task allocation in vehicular fog computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4150–4161, Jun. 2019.

[29] Q. Qi, J. Wang, Z. Ma, H. Sun, Y. Cao, L. Zhang, and J. Liao, "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning Approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4192–4203, May 2019.

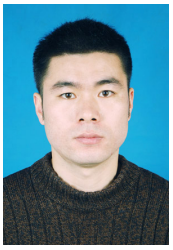
[30] A. Ciupe, B. Orza, C. Florea, and A. Vlaicu, "Skill-oriented priority scheduling for solving the resource constrained project scheduling problem," in *Proc. of IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, Sep. 2015, pp. 85–92.

[31] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan, "Mixed-integer nonlinear optimization," *Acta Numerica*, vol. 22, pp. 1–131, May 2013.

[32] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint offloading and resource allocation in vehicular edge computing and networks," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, Dec. 2018, pp. 1–7.

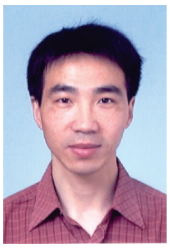
[33] K. Li, "Computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing," *IEEE Transactions on Sustainable Computing*, pp. 1–1, 2019.

[34] R. P. Dick, D. L. Rhodes, and W. Wolf, "Tgff: task graphs for free," in *Proc. of the Sixth International Workshop on Hardware/Software Codesign. (CODES/CASHE'98)*, Mar. 1998, pp. 97–101.



Yujiong Liu received the M.S. degree from Chongqing University, China, in 2010. He is currently pursuing the Ph.D. degree at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include edge computing and Internet of vehicles.

Shangguang Wang received his Ph.D. degree at Beijing University of Posts and Telecommunications in 2011. He is currently a professor and deputy director at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He has published more than 100 papers, and played a key role at many international conferences, such as general chair and PC chair. His research interests include service computing, cloud computing, and mobile edge computing. He is a senior member of the IEEE, and the Editor-in-Chief of the International Journal of Web Science.



Qinglin Zhao received the Ph.D. degree from the Chinese Academy of Sciences, Beijing, China, in 2005. From May 2005 to August 2009, he did research with The Chinese University of Hong Kong, Shatin, Hong Kong, and then with The Hong Kong University of Science and Technology, Sai Kung, Hong Kong. Since September 2009, he has been with the Faculty of Information Technology, Macau University of Science and Technology, Macau, China. He is currently a professor at the Macau University of Science and Technology. His research interests include wireless communications and networking, next generation wireless LANs, software defined wireless networking, Internet of things, Cloud/Fog computing, Blockchain and cryptocurrency.

Shiyu Du received her B.S. degree from Beijing University of Posts and Telecommunications in 2017. She is currently pursuing the M.S. degree at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. Her research interests include mobile edge computing and Internet of vehicles.

Ao Zhou received her Ph.D. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2015. She is currently an associate professor with State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She has published 20+ research papers. She played a key role at many international conferences. Her research interests include cloud computing and edge computing.

Xiao Ma received her Ph.D. degree in Department of Computer Science and Technology from Tsinghua University and B.S. degree in Telecommunication Engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2018 and 2013, respectively. She is currently a postdoctoral fellow at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. From October 2016 to April 2017, she visited the Department of Electrical and Computer Engineering, University of Waterloo, Canada. Her research interests include task scheduling, resource deployment and allocation in mobile cloud computing and mobile edge computing.

Fangchun Yang received his Ph.D. degree in communication and electronic system from the Beijing University of Posts and Telecommunication in 1990. He is currently a professor at the Beijing University of Posts and Telecommunication, China. His research interests include network intelligence and communications software. He is a fellow of the IET.