LARS: A Latency-aware and Real-time Scheduling Framework for Edge-enabled Internet of Vehicles

Shihong Hu, Guanghui Li, and Weisong Shi, *Fellow, IEEE*

Abstract—With the development of Internet of Things and mobile computing, the explosive proliferation of latency-sensitive applications raises high computation demands for mobile devices. To this end, offloading computation of applications to edge-enabled Internet of Vehicles (IoV) has emerged as an effective solution. However, most of the existing studies on this issue assume that IoV can be easily formed in the practical environment, and neglect the dependency relationship between tasks of the offloading application. In this paper, we first give several observations based on the analysis results of the real traffic dataset to verify the feasibility of aggregating vehicular resources in the real world. Then, we design a Latency-aware Real-time Scheduling Framework for the edge-enabled IoV, named LARS, in which mobile users can offload applications to LARS, and the offloading tasks can be scheduled to the appropriate vehicular resources in real-time. First, we propose a clustering-based algorithm to generate *Herds*, which treats connected vehicles as edge computation resources to provide cooperative computing services. Second, considering the dependency relationship between tasks in the job, we present a greedy-based task scheduling algorithm for offloading jobs, the objective of which is to minimize the total latency of the job as well as maximize the resource utilization of *Herds*. The simulation experiment based on the real traffic dataset shows that *Herds* generated by the proposed clustering-based algorithm can maintain a stable period to provide computing service, and the experiments on testbed include two case studies demonstrate that the superiority of the proposed scheme compared to baselines, in terms of latency and resource utilization.

Index Terms—Computation Offloading, Task Scheduling, Internet of Vehicles (IoV), Edge Computing.

1 Introduction

With the development of mobile communication technology, the rapid proliferation of high-performance intelligent terminals makes smart applications increase at a surprising rate. 4G promotes information integration and industry convergence, and intelligent applications are more widely used in industries such as finance, education, transportation, medical care, and entertainment. However, the ever-increasing demand for computing and communication of applications makes it difficult for intelligent terminals with limited resources to meet the quality of experience. In the beginning, traditional cloud computing has been widely regarded as a feasible paradigm [1]; however, the crowded transmission and long latency make it hard to meet the needs of modern applications. Fortunately, edge computing [2], [3] has emerged as a promising paradigm. In recent years, academia and industry have done a lot of research and practical applications on edge computing [4], [5], [6], [7], [8]. Undoubtedly, edge computing has efficiently solved the limitations of latency, bandwidth, energy consumption, security in traditional cloud computing [2]. However, to establish an edge computing platform for mobile users, what needs attention is that we have to deploy and maintain a tremendous number of high-cost edge servers in a large-scale area, which inevitably leads to large capital expenditures and business expenditures. Also, the intensive deployment of edge servers will lead to idle and waste of resources during off-peak hours, considering the user's dynamic time-varying demand for resources.

The emerging concept of Internet of Vehicles (IoV) [9] is employed to collect real-time traffic conditions for transportation control systems. Remarkably, the powerful onboard computer, high-capacity storage, and more advanced communication module configuration make future vehicles more intelligent. Hence, the idea of edge-enabled IoV is to treat connected vehicles as edge computation resources to form a cooperative computing cluster and make full use of the tremendous resources that are under-utilized. Furthermore, without deploying additional servers, the concept of edge-enabled IoV can also efficiently alleviate the congestion of the network during peak time. For example, Zhang *et al.* [10] exploited the fog capability of parked vehicles to help the latency-sensitive computing services. Therefore, the edge-enabled IoV as a new paradigm is an excellent complement to traditional edge computing.

Although there have been a large number of studies on IoV [11], [12], [13], [14], most of them are based on the assumption that IoV can be easily formed in the practical environment, ignoring the feasibility of IoV in the real world. Given such an observation, we evaluate the feasibility of IoV formation in a realistic environment by analyzing the real-world vehicle dataset. In this paper, we consider an edge-

- *S. Hu and G. Li are with the School of Artificial Intelligence and Computer, Jiangnan University, Wuxi, Jiangsu, 214122, China, and S. Hu is also with the Department of Computer Science, Wayne State University, Detroit, MI 48202, and G. Li is also with the Research Center for IoT Technology Application Engineering (MOE), Wuxi, Jiangsu, 214122 China. (e-mail: jnuhsh@ 163.com, ghli@jiangnan.edu.cn) (Corresponding author: Guanghui Li.)*
- *W. Shi is with the Department of Computer Science, Wayne State University, Detroit, MI 48202. (e-mail: weisong@wayne.edu)*

enabled IoV in a fixed geographical region, composed of several dynamic *Herds*. Besides, the mobility of vehicles and the dynamic variability of resources make when and where to form *Herds* in an edge-enabled IoV become a challenging issue. Therefore, we design an edge computing framework for the edge-enabled IoV, which can form *Herds* periodically and provide computing service for latency-aware applications dynamically. In the previous research on computation offloading of applications, most of the applications were simulated as a single task or multiple unrelated tasks [15], [16], [17]. However, in practical applications, the data relationship between tasks can not be ignored. For example, in an object recognition application, object detection can be performed after the feature extraction task is completed. Therefore, we considered the task dependencies between tasks while designing the task scheduling algorithm. Moreover, most works focus on optimizing the offload strategy itself based on the assumption that computing is offloaded directly to the edge. It is worth pondering whether offloading computing from users to the edge will achieve performance gains, such as reducing latency. In addition, applications from users are continuously generated and dynamically submitted to the edge-enabled IoV for real-time or near-real-time processing. Naturally, fast scheduling response is needed to keep up with the dynamic environment, which further exacerbates the difficulty of task scheduling in real-time systems.

Accordingly, the aforementioned observations and challenges motivate us to design an edge computing framework for IoV, named LARS, aiming to form *Herds* periodically to collaborate to provide computing service for applications. First, we use the realistic traffic dataset of Shenzhen city to analyze the feasibility of forming *Herds* in a practical environment. Then, based on the mobility and resource state of vehicles, we employ a classic clustering algorithm K-means to generate *Herds*, named *GetHerds*. In LARS, an application from the mobile user need to be partitioned into several tasks to generate a task dependency graph. Considering the characteristics of different tasks and the real environment, we design an accelerated evaluation to form an offloading job. To reduce the latency of jobs and maximize the resource utilization of *Herds*, we propose a Greedy-based Task Scheduling Algorithm, namely *GBTSA*, to schedule tasks to the appropriate workers and sort the tasks on each worker. The main contribution of this study can be summarized as follows:

- The feasibility of forming *Herds* in an edge-enabled IoV is verified by analyzing the real-world traffic dataset. The analysis results summarize the peak time of the day and the hot spots where vehicles gathered, which provides strong realistic conditions and a basis for the design of the LARS platform.
- In one edge-enabled IoV, according to the mobility state of vehicles such as speed, location, and trajectory, as well as the resource state such as resource amount and available time of resource, we propose a clustering algorithm *GetHerds* to generate *Herds* to coordinate to provide computing service.
- LARS, as an edge computing platform, is designed to manage the *Herds* in the covered region of an edge-enabled IoV and decide whether the task should be offloaded to *Herds*. Besides, we also propose a task scheduling algorithm called *GBTSA* for offloading jobs, the objective of which is to minimize the total latency of job as well as maximize the resource utilization of *Herds*.
- We conduct a simulation experiment based on the real traffic dataset, and the results show that *Herds* generated by *GetHerds* can maintain a stable period to provide computing service. To evaluate the performance of *GBTSA*, we build the testbed and conduct two case studies in a real system. The experimental results demonstrate that the proposed *GBTSA* is superior to baselines in terms of reducing the latency of and maximizing resource utilization.

The remainder of this paper is organized as follows. Section 2 gives a comprehensive review of related work. In Section 3, we present the observation of data analysis based on a real-world dataset and introduce the system model. Section 4 elaborates on the design of the LARS framework. The specific dynamic scheduling includes *GetHerds* and *GBTSA* are illustrated in Section 5. Section 6 provides the results of the simulation experiment, as well as the experiments on the testbed, including two case studies. Finally, Section 7 concludes our work.

2 Related work

Cyber foraging [18] coined by Satyanarayanan is a pervasive computing technique where resource-poor mobile devices offload some of their heavy work to stronger surrogate machines in the vicinity, which is the origin of computation offloading. Since then, computation offloading in cloud computing [19], [20] and mobile computing [21], [22] has started to affect the industry. In recent years, to cope with the proliferation of real-time applications, such as video analytics [23] and autonomous driving [24], computation offloading in edge computing has attracted widespread attention. As a new computing paradigm, edge computing pushes the computing from the centralized cloud to the decentralized edges, close to users, and thus effectively reduces communication latency [3] and network traffic of the core network. By deploying a large number of edge infrastructures, such as base stations (BSs), roadside units (RSUs), cloudlets [25], micro data centers (DCs), etc., to form a hierarchical computing architecture: the cloud center, the edge, and users. Therefore, offloading computation requests of latency-sensitive applications to edge-centric computing has become a hot research topic. Some studies developed the computation offloading problem as a non-linear optimization program (NLP) or a mixed-integer (MLP) program [26], [27], [28], [29]. The authors in [28] considered the energy constraints of mobile devices and dynamic network conditions to form a NLP and transformed the NLP into an equivalent integer linear program (ILP). Differently, Alameddine *et al.* [29] jointly combined the task offloading and scheduling problem into a MLP, based on the technique of the logic-based benders decomposition, they also designed a thoughtful decomposition to solve the problem. Particularly, many computing offloading strategies based on deep learning have emerged in the past two years

[30], [31], [32], [33]. The authors in [31] jointly considered server selection and data transmission mode. They employed a deep Q-learning approach to design an optimal offloading scheme, aiming to optimizing system utilities and improving offloading reliability. Besides, Huang *et al.* [32] proposed a distributed deep learning-based offloading (DDLO) algorithm to conserve energy and maintain quality of service, and they used multiple parallel DNNs to generate offloading decisions. In [33], the authors exploited the optimization problem of the content placement and content delivery in vehicular edge computing and networks. They designed a deep reinforcement learning (DRL)-based cooperative caching scheme to provide low-complexity decision making and adaptive resource management.

To compensate for the deficit resources of edge servers, edge-enabled IoV uses the existing underutilized vehicular resources in urban areas to provide computing services. It is noted that many studies use parked vehicles to enable IoVs [34], [35], [36], [37]. Li *et al.* [36] investigated a parked vehicular cloud paradigm, based on the contract; they designed an incentive mechanism to make parked vehicles contribute idle resources. Similarly, the basic idea in [34], [35], [37] is to aggregate potential vehicular resources in the parking lot to form a stationary cloud to provide users with computing resources. Moreover, many works have been done on investigating task offloading in mobile vehicular edge computing in recent years. To deal with the big data on IoVs in the smart city, the authors in [38] proposed a regional cooperative fog-computing-based intelligent vehicular network architecture, aiming to provide low-latency coordination services. Qiao *et al.* [14] presented a collaborative task offloading and output transmission mechanism for vehicular edge networks to guarantee low latency as well as application-level performance. Besides, by leveraging deep reinforcement learning, Ning *et al.* [17] built an intelligent offloading system for edge-enabled IoV, and they formulated the task scheduling and resource allocation problems to be a joint optimization problem. Differently, the work in [39] proposed an optimization model taking account of the communication and computing budgets as well as the failure probability. This is the first work to consider the offloading failure probability in heterogeneous vehicular edge computing. In summary, the studies on task offloading in IoV always can be formulated as an optimization problem, and there exist many approaches to address this problem, such as matching theory [40], [41], Lyapunov optimization [42], game theory [43], [44], [45], and deep learning-based [17], [46]. However, most of the existing works about task offloading focus on single task offloading or ignore the dependencies between tasks. Furthermore, the researchers put their attention on designing task offloading strategies based on the assumption that the realistic conditions of IoV are satisfied.

Particularly, to verify the feasibility of forming vehicular micro clouds (VMC) under practical road conditions, Higuchi *et al.* [47] firstly analyzed a realistic vehicle probe dataset of Luxembourg city, and the results showed that the connected VMCs could be formed at many locations throughout the road networks. Inspired by this work, we use the traffic dataset in Shenzhen city to analyze the feasibility of forming *Herds* and hot pot areas in the urban area. The difference is that we also design the K-means-based algorithm to generate *Herds*, which makes the foundation for the design of the LARS framework. More importantly, to minimize the total latency of the job as well as maximize the resource utilization of *Herds*, the proposed task scheduling algorithm considers the task dependency of an offloading job, as well as the dynamic real-world system conditions.

3 Observation and System Model

This section presents the observation of data analysis based on a real-world dataset and introduces the system model.

3.1 Observation

Before introducing the system model, we analyze a realistic traffic dataset of Shenzhen, China [48]. The dataset covers the traffic data of more than 10000 taxis for 24h on October 22, 2013, and collects data about every $30s$. First, we count the number of vehicles at each hour to analyze the peak time of the day. As shown in Fig. 1, from 24:00-4:00, the number

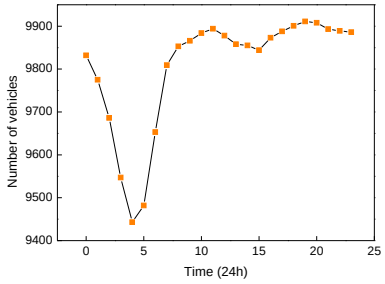


Fig. 1: The mobile vehicle distribution at different time of a day

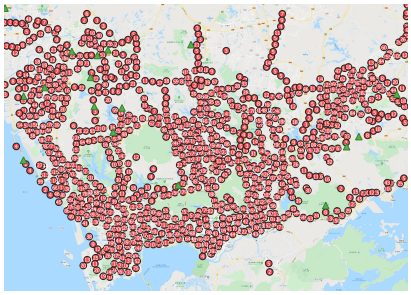


Fig. 2: Hot pots in the city during 18:00-18:05.

of active vehicles gradually decreases; this is because people are in the sleep period. From 5:00, as people start a busy day, the number of vehicles increases apparently. Furthermore, we can observe that the peak time of a day is from 18:00 to 20:00 with the largest number of active vehicles in this period, as this period is the rush time for off-work and nightlife. In this case, a large number of application requests from mobile users will be generated, and the gathered resources of mobile vehicles on the road can be used to process application requests. Then, we select one period from 18:00-18:05 to count the hot pots where the vehicles gather.

TABLE 1: Summary of Key Notation

Notation	Description	Notation	Description
M	Set of *Herds*	V	Set of vehicles equipped with edge servers
J	Set of offloading jobs from users	T_j^{arrive}	Arrive time of job j
G_j	Task dependency graph	T_j	Task set of job j
L_j	The dependency between the tasks of job j	$pr(t_i)$, $su(t_i)$	Set of precursors, successors of task t_i
$M_v(\tau)$	Mobility state of vehicle v at time slot τ	L_v	Location of vehicle v
S_v	Speed of vehicle v	ρ_v	Trajectory of vehicle v
$R_v(\tau)$	Resource state of vehicle v at time slot τ	C_v	Amount of computing resource of vehicle v
θ_v	Available time of resource in vehicle v	C_i^{acc}	Computing acceleration of a single task
I', I	Set of tasks that can/cannot be processed in parallel	$C_{I'}^{acc}$	Computing acceleration of parallel processed tasks
T_i^{local}	Local computation latency of task j	$T_i^{offloading}$	Offloading computation latency of task j
w_i	Workload of task i	r_j	Transmission rate of user j to the controller
p_j	Local processor speed of user j	p_0	Processor speed of vehicles
l	Round trip time overhead	N_{local}^{worker}	Number of workers locally
N_m^{worker}	Number of workers in *Herd* m	$C_{I',Max}^{acc}$	Maximum computing acceleration speedup
T_{tole}^j	tolerant latency of job j	P_m	Unit price of computation of *Herd* m
$N_{j',m}^{worker}$	Number of workers in *Herd* m occupied by job j	R_m	Resource utilization of *Herd* m
T_i^{star}	Ideal start time of task t_i	T_i^{end}	End time of task t_i
$T^{send}(l_{k,i})$	Time of sending the results from task t_k to task t_i	T_p^{work}	Actual working time of worker p
T_i^{pro}	Processing time of task t_i	$\hat{T}_i^{end}$	Effective end time of task t_i

TABLE 2: Number of vehicles in different hot points.

Hot point	Location(latitude, longitude)	Number of vehicles
A	(114.143951, 22.554733)	212
B	(114.062286, 22.531733)	195
C	(114.017915, 22.544110)	169
D	(114.064072, 22.531475)	162
E	(114.059547, 22.526449)	178
F	(114.278999, 22.724501)	181

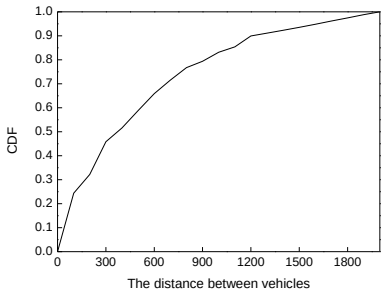


Fig. 3: The distance between vehicles.

Fig. 2 shows the aggregation of vehicles during this period, and we can see that there are lots of hot points in the entire city. In this study, a vehicle that belongs to one edge-enabled IoV at one period can contribute its idle virtualized resources to it. As we all know, the LTE [49] is more likely to be used as the communication protocol of V2I, and the effective communication range of LTE is $2km$. We randomly select different locations from the hot points in Fig. 2 and count the number of vehicles within the range of $2km$. From Table 2, we know that there are lots of vehicles within the communication range of the selected point. For example, the number of vehicles is more than 200 of point A. Moreover, the average number of vehicles within the selected hot points is about 183, which strongly verifies the feasibility of forming edge-enabled IoVs in the urban area. However, how to manage so many vehicles in one edge-enabled IoV is an intractable problem. Based on the results of Table 2, we introduce the concept of *Herd*, which consists of several vehicles with similar locations and similar resources, thus can provide computing services in cooperation with each other in one edge-enabled IoV. Considering the high-speed mobility of vehicles, the communication of V2V usually adopts the DSRC channel [50]. Generally, the communication range of DSRC is $300m$. Here, we count the distance between vehicles in the covered region of point A to verify the feasibility of forming *Herds* in an edge-enabled IoV. Fig. 3 shows the probability distributions at different distances between vehicles, and we can observe that the probability of distance within $300m$ between vehicles is about 0.45. It means that there are about 95 vehicles within $300m$ distance according to Table 2, which guarantees a sufficient number of vehicles satisfying the V2V communication to form *Herds*. As a result, how to coordinate so many vehicles in the edge-enabled IoV to form *Herds* is challenging, and we will introduce the proposed method of forming *Herds* in Section 5.

3.2 System model

We consider an edge-enabled IoV in this paper, as shown in Fig. 4. All vehicles equipped with powerful edge servers in the communication range of controller (BS or RSU) form a vehicular network, and several different vehicles aggregate into different *Herds* in one edge-enabled IoV. The controller is responsible for vehicle information collection, resource coordination, and task assignment in its edge-enabled IoV. During the peak period, once the computing request from the mobile user approaches the processing peak of the controller, the controller will periodically trigger the vehicles in the edge-enabled IoV to form several *Herds* to provide computing resources for mobile users. The details for how to form *Herds* will be illustrated in Section 5. In an edge-enabled IoV, let V denote the set of vehicles equipped with edge servers, M denote the set of *Herds*, and V_m denote the set of vehicles in *Herd* m. Mobile users in the covered region of the edge-enabled IoV can offload some computing tasks to the controller by paying for leased resources

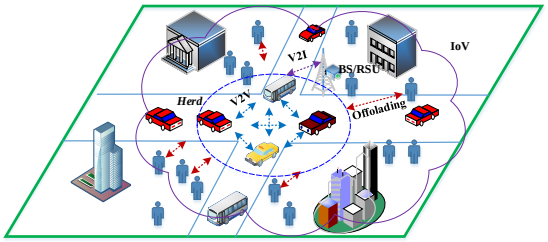
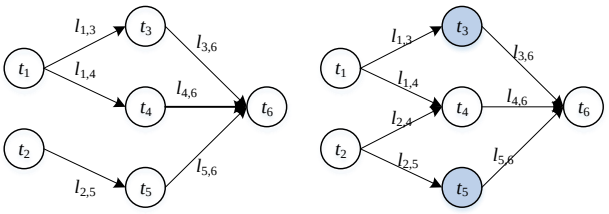


Fig. 4: System model.

to get high QoS. We use J denote the set of offloading jobs from users, and a certain job can be formulated as $J_j = \left\{t_j^{arrive}, G_j\right\}$, where t_j^{arrive} and G_j represent the arrival time and task dependency graph, respectively. Moreover, $G_j = \{T_j, L_j\}$ is modeled as a directed acyclic graph [51], [52], [53], $T_j = \{t_1, t_2, ...\}$ is the task set of job j, and L_j is the dependency between the tasks. For example, $l_{k,i} \in L_j$ means that the input data of task t_i is based on the results of task t_k; that is, t_i is the precursor of t_k . We use $pr\,(t_i)$ denote the set of precursors of task t_i, and $su\,(t_i)$ represents the set of successors of t_i. Fig. 5(a) shows a directed acyclic graph example for a job j. Job j contains six tasks, $T_j = \{t_1, t_2, ..., t_6\}$. From Fig. 5(a), we know that the precursor of t_6 is $pr\,(t_6) = \{t_3, t_4, t_5\}$ and the successor of t_1 is $su\,(t_1) = \{t_3, t_4\}$. More specifically, the task graph of Fig. 5(a) represents a traffic monitoring job. The video frames collected by roadside surveillance cameras are input into t_1 for motion detection. At the same time, the images are input into t_1 for preprocessing. Then, the results of t_2 will be sent to t_3 for pedestrian detection and t_4 for vehicle recognition. And, the results of t_2 will be sent to t_5 for plate detection. Finally, the results of t_3, t_4 and t_5 will be aggregated into t_6 for result generation. Besides, we also consider more complicated task graphs, the alternative structure. For example, as shown in Fig. 5(b), t_3 and t_5 are mutually parallel optional tasks, which represent "or" nodes and are marked in blue. Only one "or" node of the same color can be selected. Thus, in this task graph, the input of last task t_6 has two choice, one is t_3 and t_4, the other is t_4 and t_5.



(a) An instance of sequential structure of task graph.

(b) An instance of alternative structure of task graph: the blue circle represents a "or" node.

Fig. 5: Different structures of task graph.

4 LARS Framework Design

LARS is designed as an edge computing framework for edge-enabled IoVs. LARS periodically forms *Herds*, which provide computing service for latency-aware applications dynamically. The main components are applications generated by mobile users and *Herds* composed of mobile vehicles. At the peak time, when the computing demands from users are close to the controller's load, the controller will periodically schedule the vehicles within the range of the edge-enabled IoV to form several *Herds*, thereby cooperating to provide computing services for mobile users. The proposed architecture of the LARS framework is shown in Fig. 6.

4.1 Controller

As shown in Fig. 7, the controller will collect the mobility state and resource state information of all vehicles periodically. Mobility state information includes the location, the current speed, and the trajectory based on the GPS navigation system. Resource state information informs the available resource amount and the available time of the vehicle. Especially, the resource amount is represented as the number of workers in this study. Each vehicle that enters or gets out the covered region of the edge-enabled IoV will send a register or released message to the controller. Based on the collected information, the controller decides how to generate *Herds* and estimates the available virtualized resource pool.

4.2 Application

In LARS, as shown in Fig. 6, an application from the mobile user need to be partitioned as several tasks to generate a task dependency graph. The techniques of application partition are beyond the scope of this study. An accelerated evaluation needs to be performed before task offloading due to the characteristics of different tasks and the real environment. Then, the offloading controller will submit the job composed of a set of tasks to the framework by user API. According to the information of the offloading job, the job assigner first selects the most appropriate *Herd*. Then, the task scheduler calls the resource service through the internal APIs to give the optimal task provisioning message, as shown in Fig. 6 and Fig. 8. In particular, if the *Herd* resource becomes unavailable before the task is completed, the task will be migrated. In this regard, the controller of LARS must call another *Herd* to take over the task. The problem of task migration is beyond the scope of this paper.

5 Dynamic Scheduling

In this section, we give the detailed algorithm of generating *Herds*, problem formulation, and the task scheduling scheme.

5.1 Herds generator

The available resources and available time of each vehicle are very different. To coordinate vehicle resources to provide services to users, forming a *Herd* as a cluster with a similar amount of resources and similar available time can maximize resource utilization and reduce task failure and migration rates. In this study, we use the classic clustering algorithm K-means to generate *Herds*. Algorithm 1 describes the procedure of the specific process.

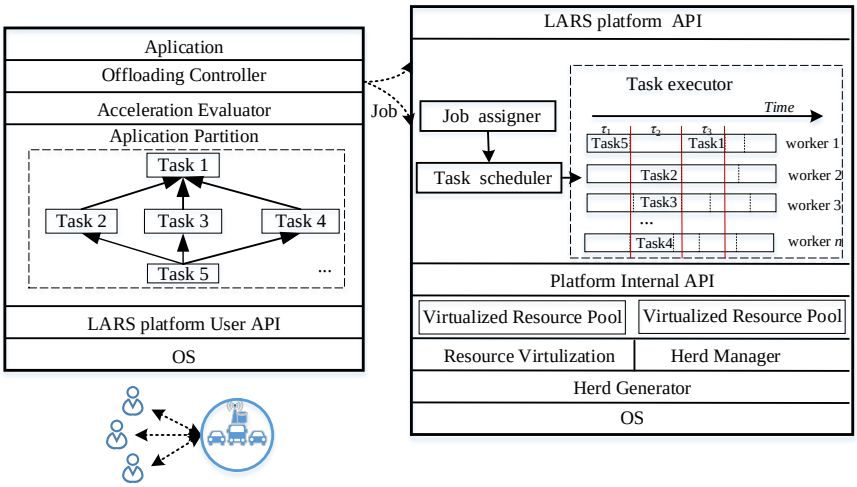


Fig. 6: The architecture of LARS framework.

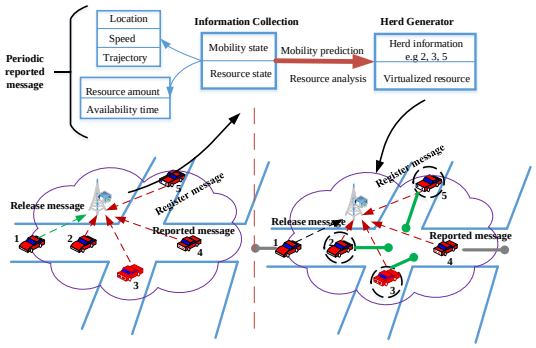


Fig. 7: *Herd* generator.

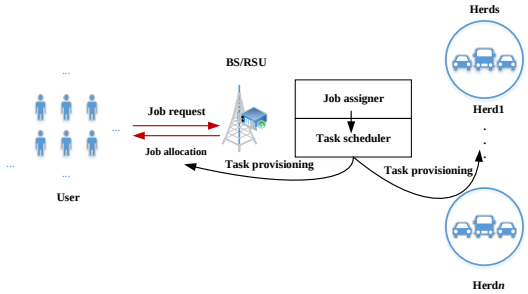


Fig. 8: Request process.

Algorithm 1 *GetHerds*

Input: Mobility state $M_v(\tau) =< L_v, S_v, \rho_v >$ and $R_v(\tau) =< C_v, \theta_v >$ of all vehicles
Output: *Herds*

1: **Procedure** *HerdGenerator*
2: **for** each vehicle $v \in V$ **do**
3: $F_v = [L_v = UpdateVehicleState(M_v, R_v), C_v, \theta_v]$
4: **end for**
5: Get the vehicle data set $D_R = [F_1, F_2, ..., F_v]$
6: $Herds(\tau) =$ *K-meansBasedCluster()*
7: **EndProcedure**
8: **Procedure** *UpdateVehicleState*
9: **for** each vehicle $v \in V$ **do**
10: $L_v = L_v(\tau - 1) + S_v \times \theta_v | \rho_v$
11: **end for**
12:
13: **for** i from 2 to k **do**
14: **for** each vehicle $v' \in V_m, v' \neq v$ **do**
15: Calculate $d_v v'$
16: **end for**
17: $u_i = \text{argmax}(d_v v'), v' = 1, 2, ..., m, v' \neq v$
18: **end for**
19: Get the k initialized Herd center $u = u_1, u_2, ..., u_k$
20: $Herds$=*K-means*(u)

The proposed ***GetHerds*** aims to generate *Herd* clusters in the edge-enabled IoV every interval time τ. Note that the input information includes the mobility state $M_v(\tau) =< L_v, S_v, \rho_v >$ and available resource state $R_v(\tau) =< C_v, \theta_v >$ of all vehicles in the edge-enabled IoV. For Procedure ***HerdGenerator***, firstly, each vehicle forms an information array F_v consisting of the location L_v, the amount of computing resource C_v, and the available time of resource θ_v (Lines 2-4). Then, the vehicle data matrix D_R composed of the information array of all vehicles has been obtained (Line 5). Based on vehicle data matrix D_R, Procedure ***HerdGenerator*** calls for Procedure ***K-meansBasedCluster*** to get the final *Herd* at time τ (Line 6). Specially, the location L_v of vehicle v needs to call for Procedure ***UpdateVehicleState*** to get the real-time location information (Lines 8-11). Procedure ***K-meansBasedCluster*** first randomly selects one cluster center and then select other $k-1$ centers based on the maximum distance among vehicles (Lines 13-21). Finally, Procedure ***K-meansBasedCluster*** uses the classic cluster algorithm k-means to generate *Herds* (Lines 22-23).

5.2 Offloading controller

Before task offloading, we designed an acceleration evaluator to evaluate whether it is necessary to offload each task in the task set T after the application partition. Computing acceleration refers to the speedup in latency of processing a task. Fig. 5 shows two instances of task graph, and we know

that some tasks have no dependencies between each other can be processed in parallel, while some tasks need to wait for all its dependent tasks to complete before they can be processed. For example, tasks 3, 4 and 5 can be processed in parallel, and task 6 can be processed after all tasks have been processed. Therefore, based on these two cases, we give two ways to evaluate the task's computing acceleration, which are expressed as (1) and (2).

$$C_i^{acc} = \frac{T_i^{local}}{T_i^{offload}} = \frac{w_i/p_j}{d_i/r_j + w_i/p_0 + l}, \tag{1}$$

$$C_{I'}^{acc} = \frac{T_{I'}^{local}}{T_{I'}^{offload}} = \frac{\sum\limits_{i \in I'} w_i/N_{local}^{worker} p_j}{d_{I'}/r_j + \max\limits_{i \in I', m \in M} w_i/N_m^{worker} p_0 + l}, \tag{2}$$

where C_i^{acc} is defined as the computing acceleration of a single task and $C_{I'}^{acc}$ represents the computing acceleration of parallel processed tasks. For Eq. (1), w_i denotes the workload of task i, d_i is the input data size of task i, p_j represents the local processor speed of user j, p_0 stands for the processor speed of vehicles, r_j is the transmission rate from user j to the controller, and l represents the round trip time overhead. For Eq. (2), I' is the set of tasks that can be processed in parallel, N_{local}^{worker} is the number of workers locally, N_m^{worker} denotes the number of workers in *Herd* m and M is defined as the set of *Herds*. Apparently, $C_i^{acc} > 1$ and $C_{I'}^{acc} > 1$ mean that the latency of local computation is greater than the total latency of the computation offloaded to the control center; that is, the computing acceleration speedup can be obtained by offloading. The offloading latency includes the transmission delay, the task processing delay, and the round trip time overhead. The acceleration evaluator will evaluate each task in the task graph of the application according to Eq. (1) and Eq. (2). Then the offloading controller will make a decision and form an offloading job $T^{off} = \{t_1, t_2, ..., t_n\}$ based on the result of the acceleration evaluator.

5.3 Job assigner

In an edge-enabled IoV, the controller manages several *Herds*. The differences in computing resources and available time of different *Herds* make the controller need to be weighed in assigning a job to the appropriate *Herd*. In this study, we assume that each job seeks to find the *Herd* with the most computing acceleration and the lowest cost. The maximum computing acceleration speedup is defined as $C_{I',Max}^{acc}$ and the total cost of leased computing resources is denoted as $T_{tole}^j P_m$, where T_{tole}^j represents the tolerant latency of job j and P_m denotes the unit price of computation of *Herd* m. The trade-off of the two objectives is investigated, and the joint objective is defined as follows:

$$P1: \max \alpha C_{I',Max}^{acc} + \frac{\beta}{T_{tole}^j P_m}, \forall m \in M, \tag{3}$$

$$s.t. N_{j,m}^{worker} = \frac{T_{tole}^j - \max\{T_j^{local}, \sum\limits_{i \in I'} \frac{w_i}{p_0}\}}{\max\limits_{i \in I} w_i/p_0}, \tag{3a}$$

$$N_{j,m}^{worker} \leq N_m^{worker}, \forall m \in M, \tag{3b}$$

$$T_{tole}^j \leq \theta_m, \forall m \in M, \tag{3c}$$

where α, $\beta \in [0, 1]$ are two scalar weights. The constraint (3a) defines the number of workers of *Herd* m $N_{j,m}^{worker}$ occupied by job j, and the constraint (3b) indicates that $N_{j,m}^{worker}$ cannot exceed the total number of workers in *Herd* m. The constraint (3c) implies the available time θ_m of *Herd* m should be bigger than the tolerant latency T_{tole}^j of job j . Problem P1 has a deterministic solution and is easy to be solved.

5.4 Task scheduling

5.4.1 Problem formulation

For a job $J_j = \left\{T_j^{arrive}, G_j\right\}$, T_j^{arrive} is the arrival time of and G_j is the task dependency graph of job j. Each task $t_i \in G_j$ needs to wait until all tasks in its precursors have been completed and sent the results to it before starting running.

Definition 1. The ideal start time of task t_i is defined as:

$$T_i^{star} = \max_{t_k \in pr(t_i)} (T_k^{end}(t_k) + T^{send}(l_{k,i})), \tag{4}$$

where $T^{end}(t_i)$ denotes the end time of task t_i, $T^{send}(l_{k,i})$ indicates the time of sending the results from task t_k to task t_i. The end time of job j is determined by the maximal finish time of its all tasks, which can be defined as:

$$T^{end}(J_j) = \max_{t_i \in T_j} (T^{end}(t_i)), \tag{5}$$

where $T^{end}(J_j)$ represents the end time of job j. For mobile users, the objective is to minimize the total completion time of an offloading job j, thus obtaining near-real-time results. The total completion time of job j is the difference between the end time and the arrival time, which is expressed as:

$$Minimize \quad T^{end}(J_j) - T_j^{arrive}. \tag{6}$$

For the computing service providers, they seek to maximize resource utilization to reduce energy-related consumption. For a *Herd*, its resource utilization refers to the actual average number of served workers.

5.4.2 Theoretical analysis

It is impossible to obtain optimal solutions because the problem of task scheduling in dynamic environments is NP-hard [54], [55]. Two key steps for task scheduling on the LARS framework, one is to map tasks to the appropriate worker, and the other is to perform a reasonable execution order of tasks on the same worker, thereby minimizing the delay of jobs and maximizing the resource utilization of *Herds*. In this study, refer to [56], we apply the minimum earliest start time first rule (MESTF) to sort the tasks on the same worker.

Theorem 1. Assume all the tasks in task set $T = \{t_1, t_2, ..., t_k\}$ are parallel and their ideal start time satisfy that $T_1^{star} \leq T_2^{star} \leq ... \leq T_k^{star}$, the total completion time of task set $T = \{t_1, t_2, ..., t_k\}$ on the same worker can be minimized by the MESTF rule.

Proof: See the Appendix.

Fig. 9 gives an example of Theorem 1, the ideal start time of tasks in task set $T = \{t_1, t_2, t_3\}$ is $T_1^{star} = 0$, $T_2^{star} = 20$ and $T_3^{star} = 35$, respectively. Besides, assume the processing time of tasks is $T_1^{pro} = 15$, $T_1^{pro} = 10$ and $T_3^{pro} = 10$. If we apply the MESTF rule to sort these tasks in worker 1,

as shown in Fig. 9(a), the processing order is $t_1 \rightarrow t_2 \rightarrow t_3$, and the total completion time is $45s$. However, if we change the order of t_2 and t_3, the total completion time is $55s$, as shown in Fig. 9(b). As a result, we can know that the processing order by the MESTF rule can effectively reduce the completion time of tasks on the same worker.

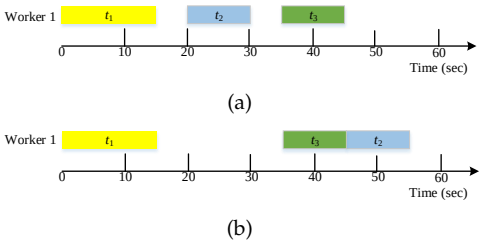


Fig. 9: An example of Theorem 1.

Theorem 2. Let $pr(t_i) = \{t_1, t_2, ..., t_m\}$ and $T_1^{end} + T^{send}(l_{1,i}) \geq T_2^{end} + T^{send}(l_{2,i}) \geq ... \geq T_m^{end} + T^{send}(l_{m,i})$, task t_i start time can be minimized as $T_{min}^{star}(t_i) = \min_{1 \leq l \leq k} \max \left\{ MT(\bigcup_{i=1}^{l} t_i), T_{l+1}^{end} + T^{send}(l_{1+1,i}) \right\}$, where $MT(\bigcup_{j=1}^{l} t_i) = \min \max_{t_i \in T'} \{T_i^{end}\}$ refers to the maximal completion time of the task set $\{t_1, t_2, ..., t_l\}$ base on MESTF rule.

Proof: See the Appendix.

Corollary. According to **Theorem 2**, we can obtain the following corollary: some tasks in the precursors $pr(t_i)$ of t_i may not be in the same worker. If there is a task t_m satisfies $T^{send}(l_{m,i}) > T_m^{pro}$, duplicating the task to the worker where t_i is located can be minimized the total completion time of t_i.

As illustrated in Fig. 10(a), the sample task graph consists of five tasks $\{t_1, t_2, t_3, t_4, t_5\}$. For task t_4, its precursors $pr(t_4) = \{t_1, t_2, t_3\}$. The time of sending results from precursors to the task is written on the arrow between them, and the processing time is also marked around the tasks. For example, the processing time of task t_3 is $T_3^{pro} = 5$ and the time sending the result from task t_3 to task t_4 is $T^{send}(l_{3,4}) = 15$. From Fig. 10(b), we know that task t_4 has to wait for all its precursors have been processed and all the results have been sent to it. Since it takes time for the result of task t_3 to be sent from worker 1 to task t_4 on worker 2, the start time of task t_4 is delayed to $40s$. As shown in Fig. 10(c), we duplicate the task t_3 to worker 2 and the start time of task t_4 can be advanced from $40s$ to $30s$.

Definition 2. The effective end time of task t_i is defined as:

$$\hat{T}^{end}(t_i) = \begin{cases} \min\limits_{t_q \in su(t_i)} \left(\max\limits_{t_k \in pr(t_q)} T^{end}(t_k) - T^{send}(l_{q,i}) \right), & su(t_i) \neq \varnothing, \\ 0, & su(t_i) = \varnothing. \end{cases} \quad (7)$$

Theorem 3. If task t_i satisfies $T_i^{end} \leq \hat{T}_i^{end}$, then the end time T_i^{end} of t_i has no effect on its successors' start.

Proof: See the Appendix.

To explain Theorem 3 effectively, we give an instance, as shown in Fig. 11. From Fig. 11, we know task t_1 and t_2 have been processed, and the end time is $60s$ and $80s$, respectively. The task t_3 is unscheduled and

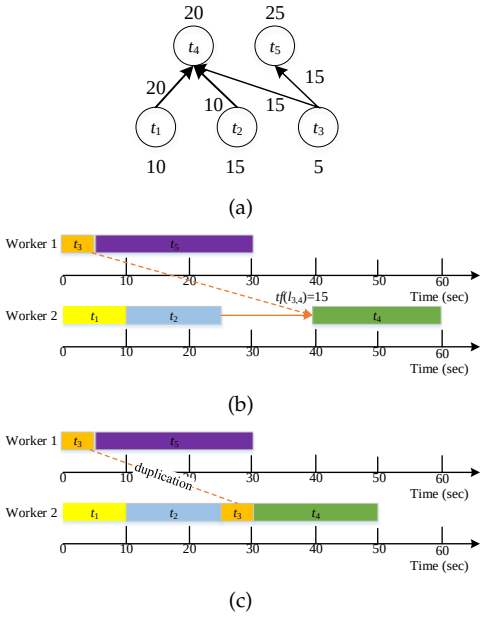


Fig. 10: An example of Corollary.

its successors are t_4 and t_5, the time of sending results from t_3 to t_4 and t_5 is $15s$ and $40s$. Based on Eq. 7, the effective end time of t_3 can be calculated as $\hat{T}_1^{end} = \min\limits_{t_q \in t_4, t_5} \left\{ \max\limits_{t_k \in pr(t_4)} -T^{send}(l_{4,3}), \max\limits_{t_k \in pr(t_5)} -T^{send}(l_{5,3}) \right\} = \min\{60 - 15, 80 - 40\} = 40$. If the task t_3 is processed and the actual end time $T_3^{end} \leq 40$, which means the task t_3 has no effect on its successor task t_4's start time.

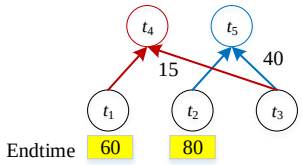


Fig. 11: An example of Theorem 3.

According to Theorem 1, Theorem 2 and Theorem 3, we design a greedy-based task scheduling algorithm, namely *GBTSA*, to schedule tasks to the appropriate workers and sort the tasks on each worker. The detailed process of *GBTSA* is described in Algorithm 2.

In the proposed *GBTSA*, $WorkerS$ denotes the set of available workers in *Herd* m and is responsible for recording the information of available workers; $WaitQ$ represents the waiting queue and is initialized to be empty before task scheduling; $ExcuS$ denotes the set of executing tasks and is also initialized to be empty. Procedure ***Preparation*** is designed to update and sort the tasks in the $WaitQ$ (Lines 3-15). When a new job arrives, all the tasks of the job will be added to $WaitQ$ and wait to be scheduled for the optimal workers. Then, the tasks with no precursors or their precursors have been scheduled will be select in $ProT$ (Line 5). For each task in $ProT$, if it has a parallel optional task t_i^{opt} and the optional task is already in the execution queue $ExcuS$, the task t_i will be deleted from $WaitQ$; if not, then the algorithm *GBTSA* calls for Procedure ***Schedule*** to find

Algorithm 2 *GBTSA*

1: $WorkerS \leftarrow$ Get all the available workers in *Herd* m
2: $WaitQ \leftarrow \varnothing$;$ExcuS \leftarrow \varnothing$
3: **Procedure** *Preparation*
4: **while** a new job arrives **do**
5: Add all the tasks of the new job into $WaitQ$
6: **while** $WaitQ! = \varnothing$ **do**
7: $ProT = \{t_i | pr(t_i) = \varnothing \cup pr(t_i) \subseteq ExcuS\}$
8: **for** each task $t_i \in ProT$ **do**
9: **if** $t_i^{opt} = \varnothing$ or $t_i^{opt} \subseteq ExcuS$ **then**
10: $[selWorker, reTaskS]$ = **Schedule**(t_i)
11: $ExcuS(selWorker) \leftarrow ExcuS \cup reTaskS \cup \{t_i\}$
12: **else**
13: delete t_i from $WaitQ$
14: **end if**
15: **end for**
16: **end while**
17: **end while**
18: **EndProcedure**
19: **Procedure** *Schedule*(t_i)
20: $selWorker \leftarrow NULL$; $reTaskS \leftarrow \varnothing$; $MinT \leftarrow \infty$
21: Compute $\hat{T}^{end}(t_i)$ of task t_i
22: **for** each worker $w_k \in WorkerS$ **do**
23: $tempRT \leftarrow \varnothing$
24: Use MESTF rule and get $T^{end}(t_i)$ of task t_i on worker w_k
25: **while** ture **do**
26: **if** $T^{end}(t_i) < MinT$ **then**
27: $selWorker = w_k$;
28: $MinT = max(T^{end}(t_i), \hat{T}^{end}t_i))$;
29: $reTaskS \leftarrow tempRT$;
30: **if** $T^{end}(t_i) < \hat{T}^{end}t_i)$ **then**
31: break;
32: **end if**
33: **end if**
34: $St \leftarrow \text{argmax}_{t_k \in pr(t_i)}(T^{end}(t_k) + T^{send}(l_{k,i}))$
35: **if** $St! = NULL$ & $St \notin tempRT$ & $St \notin ExcuS(w_k)$ **then**
36: $tempRT \leftarrow tempRT \cup \{St\}$
37: Assuming that all the tasks in $tempRT$ are re-executed on w_k
38: Update $T^{end}(t_i)$ of task t_i
39: **else**
40: break;
41: **end if**
42: **end while**
43: **end for**
44: **EndProcedure**

the optimal worker $selWorker$ and the set of tasks $reTaskS$ associated with it that need to be duplicated. Finally, the task and its $reTaskS$ will be added into $ExcuS(selWorker)$ of selected worker (Lines 8-10). Procedure ***Schedule*** gives a detailed process of task scheduling based on Theorem 1, Theorem 2 and Theorem 3. Note that $selWorker$ denotes the selected worker for task t_i, $reTaskS$ is used to record the tasks needed to be re-executed, and $MinT$ stands for the minimal end time of task t_i. Firstly, Procedure ***Schedule*** computes the effective end time of task t_i (Line 18). Then, Procedure ***Schedule*** tries to schedule task t_i to a worker satisfying $T^{end}(t_i) < \hat{T}^{end}(t_i)$ (Lines 25-26), which greatly reduces the probability of task migration. If it is infeasible, the worker with the minimal end time of task t_i will be recorded (Lines 21-24). Using the advantage of Theorem 2, Procedure ***Schedule*** strives to advance the start time of task t_i by re-executed some tasks in its precursors (Lines 16-35). Besides, the selected tasks are added into $tempRT$ and the iteration until the start time of task t_i can no longer be advanced.

6 Experiments and Evaluation

In this section, we aim to evaluate the performance of two proposed algorithms, including *GetHerds* and *GBTSA*. First, we carry on a simulation experiment for the proposed *GetHerds*. Second, we introduce the experimental hardware and environment setup and present evaluation results under two study cases for *GBTSA*.

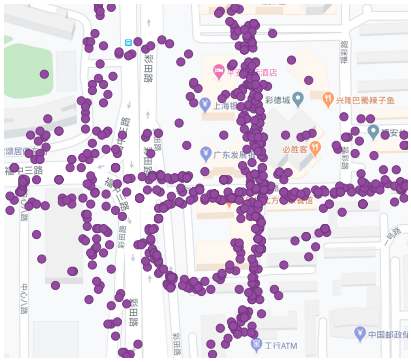


Fig. 12: The snapshot of vehicles at time 18:00.

TABLE 3: The results of 10 generated *Herds*. Acronyms used in this Table: Average Distance (AD), Differences of Speed (DoS), Differences of Resource Amount (DoRA), Differences of Available Time (DoAT).

Herd	AD	DoS	DoRA	DoAT
1	297	3.8678	1.1662	7.9095
2	286	3.2619	0.8944	6.2097
3	231	3.7202	1.1662	5.831
4	255	2.0591	0.8	10.4881
5	272	4.4091	0.4899	6.7646
6	210	3.9699	1.0198	6.5605
7	283	2.7568	0.4899	8.7316
8	266	2.4166	0.9798	6.5238
9	247	2.5612	0.4899	8.6348
10	295	4.4452	0.7483	8.0895

6.1 Simulation Results

In this simulation, we use the real traffic dataset of Shenzhen city introduced in Section 3. We select one point (114.143951, 22.554733) as the controller location, and the communication between controller and vehicle is assumed to be LTE. Within a $2km$ covered region of the controller, all the snapshot locations of vehicles, as well as speed data at time 18:00 on October 22, 2013, are used for simulation, as shown in

TABLE 4: The average results of different number of *Herds*. Acronyms used in this Table: Average Distance (AD), Average Differences of Speed (ADoS), Average Differences of Resource Amount (ADoRA), Average Differences of Available Time (ADoAT)

Number of *Herds*	AD	ADoS	ADoRA	ADoAT
10	264	3.3467	0.8244	7.5743
15	252	2.8122	0.6421	6.9856
20	250	2.7723	0.6678	5.7342
30	244	2.4124	0.7012	6.3251
40	239	2.4781	0.5908	6.6624
50	232	2.2912	0.6186	5.5929

Fig. 12. In this paper, the resource state of vehicles is randomly generated, the range of computing resource amount C_v is $[4, 6]$, and the range of available time of resource AP_v is $[3, 5]\, min$. To evaluate the stability and cohesiveness of generating *Herds* by *GetHerds*, we use Euclidean distance to evaluate the state similarity of all vehicles in each *Herd* in terms of speed, resource amount, and available time. Table 3 shows the results of the selected 10 generated *GetHerds*. From Table 3 and Table 4, we can see the average distance between vehicles of each *Herd* is all below $300m$, which satisfies the V2V communication condition. The speed of the real vehicle at peak time is distributed between $20km/h$ and $40km/h$. The third column of Table 3 shows that the speed difference of one *Herd* is small and Table 4 indicates that the same observation, and most of the *Herds* are around 3. Besides, the differences between resource amount and the available time of most *Herds* are below 1 and 10, respectively, which are shown in the fourth and fifth columns of Table 3 and Table 4. In summary, the mobility and resource state of each generated *Herds* are similar and can maintain a stable period to provide computing service.

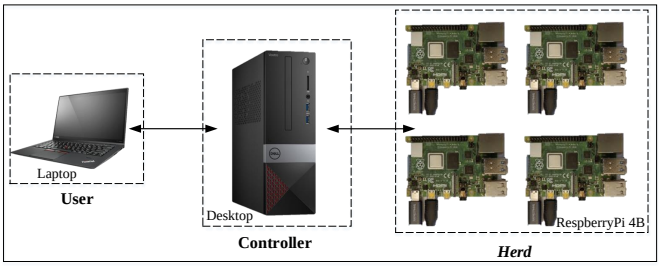


Fig. 13: LARS system implementation .

6.2 Experiment on Testbed

6.2.1 Evaluation Setup

The main purpose of the experiment in this subsection is to verify the performance of the scheduling algorithm *GBTSA*. Thus, we used four edge nodes, one desktop, and one laptop to build the experimental testbed for LARS. In our experiment, edge nodes are assumed to be the *Herd* with a stable and available state, and each edge node is considered as one computing worker of the *Herd*, the desktop is responsible for resource scheduling and management as a controller, and the laptop is used to generate applications to decide whether or not offload to the LARS framework. The four edge nodes are donated from Intel with the same hardware specifications. All devices and edge nodes are connected by socket communication. We use the Python SimPy tool to simulate the Poisson arrival process of jobs. And, the arrival rate λ is randomly generated from $[0.2, 0.5]$, e.g., if $\lambda = 0.2$, the expected number of jobs that reach the system per second is $1/\lambda = 5$. Table 5 gives the specific information of all hardware used in this evaluation. In this paper, we employed two real-world applications to verify the performance of *GBTSA*. In order to prevent the occurrence of task migration, we assume that *Herd*'s stable working time is greater than the time for all tasks to be processed.

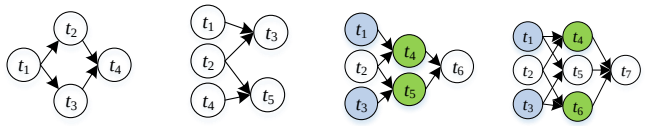
TABLE 5: Hardware Setup

Platform	DESKTOP	LAPTOP	RaspberryPi 4B
CPU	Inter i7-4770	Inter i7-5600U	ARM Cortex-A72
Socket	1	1	1
Architecture	x86,64	x86,64	ARM
Memory	16	8	4
Disk	1TB SSD	256G SSD	64G SD

6.2.2 Baselines

We compare the performance of *GBTSA* against the following three baselines,

- **Local**: it means all the tasks of jobs are processed locally.
- **Random**: the controller will select a worker of *Herd* randomly to process the offloaded task.
- **Shortest Transmission Time First (STTF) [57]**: it means that the controller tends to schedule tasks on the worker that has the shortest estimated latency to transfer the tasks. The controller maintains a table to record the latency of transmitting data to each available worker.
- **Shortest queue Length First (SQLF) [57]**: it means that the controller tends to schedule tasks to the worker which has the least number of tasks queued upon the time of query.



(a) SS: 4 tasks (b) SS: 5 tasks (c) AS: 4 tasks (d) AS: 5 tasks

Fig. 14: Different task dependency graph: Sequential structure (SS) or Alternative structure (AS).

6.2.3 Case study 1: Scene Understanding

The scene understanding application can help users grasp the information of the target environment to make a reasonable judgment. We use ADE20K dataset [58] provided by MIT for scene understanding experiments. We assume that four or five scene understanding tasks form a job, and the dependency of these tasks is predefined in advance. We

design two types of task graphs for jobs: sequential structure and alternative structure to impose the task dependency for jobs. Fig. 14 shows the designed task dependency graphs. A job with an alternative task structure means that the task chain to complete the job is optional. As shown in Fig. 14(c) and (d), the blue and green circle nodes are "or" nodes, which means that only one "or" node of the same color can be selected. For example, in Fig. 14(c), we can select t_1 and t_4 or t_3 and t_5 for a job. Particularly, jobs are generated according to Poisson distribution with the arrival rate λ, and the task dependency graph of each job is randomly selected from Fig. 14.

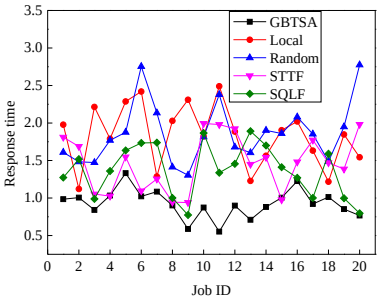
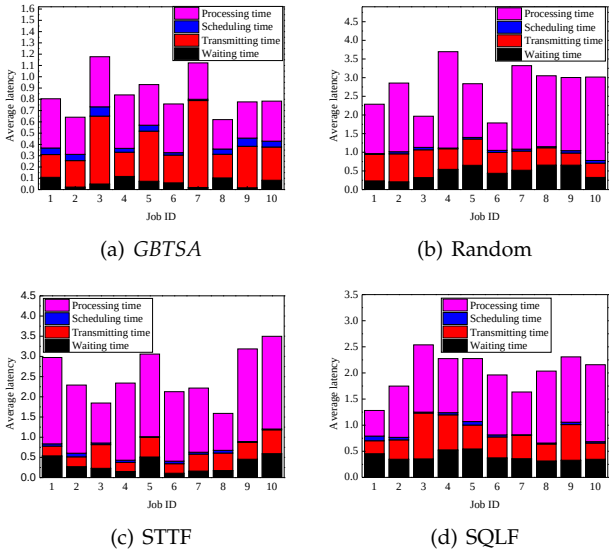


Fig. 15: Case 1: Response time vs. 20 different jobs.

Application side: From the application side, the objective of the proposed *GBTSA* is to minimize the total latency of an offloading job. In this experiment, the jobs generated by users follow the Poisson distribution, and the arrival rate λ is randomly generated from $[2, 5]$. To show the superiority of the proposed *GBTSA*, we conduct comparative experiments with baselines. First, we compare the response time of jobs under different schemes. Fig. 15 shows the results of 20 jobs under five schemes: the proposed *GBTSA*, local computing, random scheduling, STTF and SQLF. From Fig. 15, in general, we can observe that the response time of jobs under *GBTSA* is the lowest, followed by SQLF and STTF, then random scheduling, and the response time of jobs is the highest when computed locally. Apparently, the limited computing resources extend the local latency of jobs. Besides, STTF and SQLF perform similarly; overall, SQLF works better than STTF. The STTF scheme tends to schedule a task to the worker with the least transmitting time. Sometimes a lower total latency will be achieved, but it may cause a worker to saturate and increase the response time. Also, the SQLF scheme has a similar problem. Scheduling a task to the worker with the shortest queue without considering the transmission latency may lead to an excessive delay in transmission and increase the response time. We can see that the response time of job 6 and job 20 is abnormally high, and this is because the randomness of scheduling may schedule some tasks to busy workers and thus extending the waiting time in queue. In contrast, the total latency of each job under *GBTSA* remains stable.

Generally, the response time of a job consists of transmitting time, scheduling time in the controller, waiting time in queue, and processing time on a worker. To investigate which latency affects the total latency the most, we counted the different latency of each job. As shown in Fig. 16(a), for *GBTSA*, the transmitting time and processing time account



(a) *GBTSA* (b) Random

(c) STTF (d) SQLF

Fig. 16: Case 1: Different latency vs. 10 different jobs.

for the largest latency and waiting time also occupies a small part. The scheduling time in the control center is very short. It is worth noting that the transmitting time is a bottleneck factor that causes the total delay, and it will be the focus of our future research work. Fig. 16(b) shows the different latency results of 10 jobs under random scheduling; apparently, the processing time of jobs is the largest part of the total latency. This is because improper scheduling makes task processing time longer, as well as waiting time. Fig. 16(c) and Fig. 16(d) show the different latency results of 10 jobs under STTF and SQLF, respectively. Like the previous analysis, STTF has an advantage in transmitting time compared to SQLF, but SQLF is superior in waiting and processing time. In general, observing the range of the y-axis of Fig. 16, we can find that the average latency of GBTSA is smaller than the other three scheduling schemes.

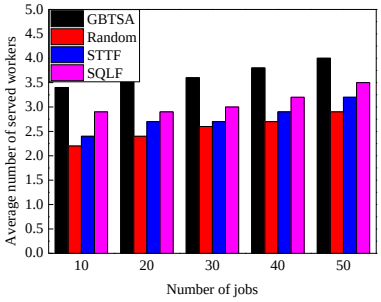


Fig. 17: Case 1: Average workers vs. different number of jobs.

***Herd* side:** From the *Herd* side, the objective is to maximize resource utilization. We use the average served workers to evaluate the resource utilization of different scheduling schemes. From Fig. 17, we can observe that under *GBTSA*, no matter how many jobs, the number of served workers is almost 4. This means that almost all workers participate in the processing of the job. Followed

by SQLF, the average number of served workers is almost 3, which indicates that the SQLF scheme has a relatively good performance in resource utilization. However, the average number of served workers of STTF and random schemes is less than 3, which means only no more than 3 workers process the job, resulting in low resource utilization. The randomness of the random scheme and the single tendency of the STTF scheme for the shortest transmitting time make them less realizable in terms of resource utilization.

6.2.4 Case study 2: Driving Behavior Detection

The second case study is driving behavior detection, where security monitors in a traffic sensing application use captured vehicular images to detect dangerous behavior. For example, a surveillance camera at the intersection captures the driver on the phone while driving, and then alerts the dangers signal to traffic police. We use Berkeley DeepDrive dataset [59] for driving behavior detection experiments. Similar to case study 1, four or five tasks are assumed to be one job, and the task dependency graph of each job is randomly selected from Fig. 14. Besides, the job arrival process also obeys the Poisson distribution with the arrival rate λ.

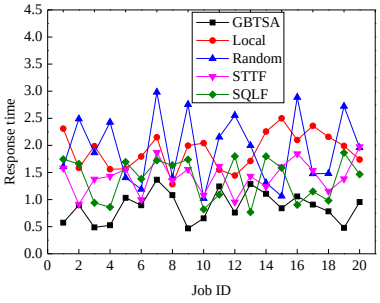
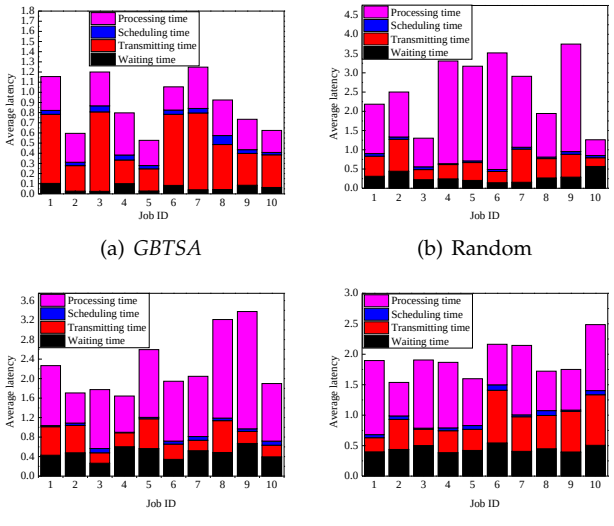


Fig. 18: Case 2: Response time vs. 20 different jobs.



(a) *GBTSA* (b) Random

(c) STTF (d) SQLF

Fig. 19: Case 2: Different latency vs. 10 different jobs.

Application side: In this experiment, we use the Python SimPy tool to simulate the job arrival process, and the arrival rate λ is randomly generated from $[2, 5]$. Fig. 18 shows the response time results of 20 jobs under five schemes. Generally, *GBTSA* still maintains an advantage in reducing the response time among five schemes. However, we also observe that the response time of a few jobs under baselines will be lower than the offloading, which may be due to unstable communication. In summary, *GBTSA* can keep good performance in reducing the response time compared to baselines in both applications, where exit many uncertainties in the real system. Further, Fig. 19 shows the different latency results of 10 jobs under four scheduling schemes. In this case study, the transmitting time occupies the largest proportion of the total latency for *GBTSA*, followed by the processing time, as shown in Fig. 19(a). In contrast, for random scheduling in Fig. 19(b), the processing time accounts for the largest part of latency. Also, as shown in Fig. 19(c), the STTF scheme has a similar problem. The SQLF scheme also shows good performance in reducing processing time, but the transmitting time accounts for more. Based on these observations, we should realize that communication between users and *Herds* has a great impact on job completion. The stable and fast communication method is an important technology in edge computing.

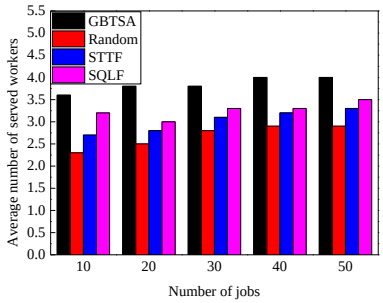


Fig. 20: Case 2: Average workers vs. different number of jobs.

***Herd* side:** Similarly, we employ resource utilization to compare the performance of these four scheduling schemes. As shown in Fig. 20, we can see that the average served workers of *GBTSA* are almost 4 under the different number of jobs. Followed by SQLF, the average number of served workers is bigger than 3. In this case study, the gap between STTF and SQLF is not very large, and the average number of served workers under STTF remains at 3. However, for random scheduling, the number of served workers is less than 3. As a result, fewer served workers result in lower resource utilization. In conclusion, the proposed *GBTSA* scheme can keep high resource utilization of *Herd*.

7 Conclusion

In this paper, according to observations based on the analysis results of the real traffic dataset in Shenzhen, we verify the feasibility of forming *Herds* in an edge-enabled IoV. To better manage the computing resources in the vehicular network and provide computing services effectively, we design a dynamic scheduling framework LARS. In LARS, a K-means-based clustering algorithm *GetHerds* is proposed to generate *Herds*. Particularly, the task of an application

should be evaluated to determine whether it can benefit from offloading to *Herds*. For offloading jobs, the job assigner in the controller first selects the appropriate *Herd*, and then a greedy-based task scheduling algorithm *GBTSA* is used to schedule tasks to the appropriate workers and sort the tasks on each worker. The simulation experiment based on the real traffic data shows that *Herds* generated by *GetHerds* can maintain a stable period to provide computing service. The experiments on the testbed include two case studies that show that *GBTSA* has a good performance in reducing the total latency of jobs, as well as maximizing the resource utilization of the edge-enabled IoV. The experimental results from a testbed reveal that the communication time is a bottleneck factor resulting in the total latency of jobs. Based on this observation, we will build an experimental environment based on 5G communications to make computation offloading of applications to the IoV more realistic.

ACKNOWLEDGMENTS

This work was supported in part by the scholarship from China Scholarship Council, in part by the National Natural Science Foundation of China (No. 62072216), Jiangsu Agriculture Science and Technology Innovation Fund (No. CX(19)3087), Wuxi International Science and Technology Research and Development Cooperative Project (No. CZE02H1706), the 111 Project (B12018).

REFERENCES

[1] R. Hussain, F. Abbas, J. Son, and H. Oh, "TIaaS: Secure cloud-assisted traffic information dissemination in vehicular ad hoc networks," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 2013, pp. 178–179.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] W. Shi, G. Pallis, and Z. Xu, "Edge computing [scanning the issue]," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1474–1481, 2019.

[4] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[5] B. I. Ismail, E. M. Goortani, M. B. Ab Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe, "Evaluation of docker as edge computing platform," in *2015 IEEE Conference on Open Systems (ICOS)*. IEEE, 2015, pp. 130–135.

[6] 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, 2017.

[7] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, "Integrated blockchain and edge computing systems: A survey, some research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1508–1532, 2019.

[8] O. Krestinskaya, A. P. James, and L. O. Chua, "Neuromemristive circuits for edge computing: A review," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[9] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *2014 IEEE world forum on internet of things (WF-IoT)*. IEEE, 2014, pp. 241–246.

[10] Y. Zhang, C.-Y. Wang, and H.-Y. Wei, "Parking reservation auction for parked vehicle assistance in vehicular fog computing," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3126–3139, 2019.

[11] W. Xu, H. Zhou, N. Cheng, F. Lyu, W. Shi, J. Chen, and X. Shen, "Internet of vehicles in big data era," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 1, pp. 19–35, 2017.

[12] M. Chen, Y. Tian, G. Fortino, J. Zhang, and I. Humar, "Cognitive internet of vehicles," *Computer Communications*, vol. 120, pp. 58–70, 2018.

[13] Z. Ning, X. Hu, Z. Chen, M. Zhou, B. Hu, J. Cheng, and M. S. Obaidat, "A cooperative quality-aware service access system for social internet of vehicles," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2506–2517, 2017.

[14] G. Qiao, S. Leng, K. Zhang, and Y. He, "Collaborative task offloading in vehicular edge multi-access networks," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 48–54, 2018.

[15] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[16] N. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/aerial-assisted computing offloading for iot applications: A learning-based approach," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1117–1129, 2019.

[17] Z. Ning, P. Dong, X. Wang, J. J. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 6, pp. 1–24, 2019.

[18] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17, 2001.

[19] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2014.

[20] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE transactions on parallel and distributed systems*, vol. 26, no. 12, pp. 3317–3329, 2014.

[21] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[22] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *International Conference on Mobile Computing, Applications, and Services*. Springer, 2010, pp. 59–79.

[23] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 377–392.

[24] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[25] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[26] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 365–375.

[27] T. T. Nguyen, L. Le, and Q. Le-Trung, "Computation offloading in mimo based mobile edge computing systems under perfect and imperfect csi estimation," *IEEE Transactions on Services Computing*, 2019.

[28] S. Misra and N. Saha, "Detour: dynamic task offloading in software-defined fog for iot applications," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1159–1166, 2019.

[29] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 668–682, 2019.

[30] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[31] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7635–7647, 2019.

[32] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile Networks and Applications*, pp. 1–8, 2018.

[33] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, and N. Ansari, "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 247–257, 2019.

[34] W. He, G. Yan, and L. Da Xu, "Developing vehicular data cloud services in the iot environment," *IEEE transactions on industrial informatics*, vol. 10, no. 2, pp. 1587–1595, 2014.

[35] E. Al-Rashed, M. Al-Rousan, and N. Al-Ibrahim, "Performance evaluation of wide-spread assignment schemes in a vehicular cloud," *Vehicular Communications*, vol. 9, pp. 144–153, 2017.

[36] C. Li, S. Wang, X. Huang, X. Li, R. Yu, and F. Zhao, "Parked vehicular computing for energy-efficient internet of vehicles: A contract theoretic approach," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6079–6088, 2018.

[37] T. Kim, H. Min, and J. Jung, "Vehicular datacenter modeling for cloud computing: Considering capacity and leave rate of vehicles," *Future Generation Computer Systems*, vol. 88, pp. 363–372, 2018.

[38] W. Zhang, Z. Zhang, and H.-C. Chao, "Cooperative fog computing for dealing with big data in the internet of vehicles: Architecture and hierarchical resource management," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 60–67, 2017.

[39] K. Xiong, S. Leng, C. Huang, C. Yuen, and Y. L. Guan, "Intelligent task offloading for heterogeneous v2x communications," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[40] F. Chiti, R. Fantacci, and B. Picano, "A matching theory framework for tasks offloading in fog computing for iot systems," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5089–5096, 2018.

[41] H. Liao, Z. Zhou, X. Zhao, B. Ai, and S. Mumtaz, "Task offloading for vehicular fog computing under information uncertainty: A matching-learning approach," in *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE, 2019, pp. 2001–2006.

[42] L. Pu, X. Chen, J. Xu, and X. Fu, "D2d fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3887–3901, 2016.

[43] M. Liu and Y. Liu, "Price-based distributed offloading for mobile-edge computing with computation capacity constraints," *IEEE Wireless Communications Letters*, vol. 7, no. 3, pp. 420–423, 2017.

[44] Z. Xiong, J. Kang, D. Niyato, P. Wang, and V. Poor, "Cloud/edge computing service management in blockchain networks: Multi-leader multi-follower game-based admm for pricing," *IEEE Transactions on Services Computing*, 2019.

[45] D. Zhang, Y. Ma, C. Zheng, Y. Zhang, X. S. Hu, and D. Wang, "Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 243–259.

[46] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, 2017.

[47] T. Higuchi, J. Joy, F. Dressler, M. Gerla, and O. Altintas, "On the feasibility of vehicular micro clouds," in *2017 IEEE Vehicular Networking Conference (VNC)*. IEEE, 2017, pp. 179–182.

[48] Y. Yang, X. Xie, Z. Fang, F. Zhang, Y. Wang, and D. Zhang, "Vemo: Enabling transparent vehicular mobility modeling at individual levels with full penetration," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

[49] S. Chen, J. Hu, Y. Shi, and L. Zhao, "Lte-v: A td-lte-based v2x solution for future vehicular network," *IEEE Internet of Things journal*, vol. 3, no. 6, pp. 997–1005, 2016.

[50] C. Zhu, J. Tao, G. Pastor, Y. Xiao, Y. Ji, Q. Zhou, Y. Li, and A. Ylä-Jääski, "Folo: Latency and quality optimized task allocation in vehicular fog computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4150–4161, 2018.

[51] J. Wang, D. Crawl, I. Altintas, and W. Li, "Big data applications using workflows for data parallel computing," *Computing in Science & Engineering*, vol. 16, no. 4, pp. 11–21, 2014.

[52] L. Wang, Y. Ma, J. Yan, V. Chang, and A. Y. Zomaya, "pipscloud: High performance cloud computing for remote sensing big data management and processing," *Future Generation Computer Systems*, vol. 78, pp. 353–368, 2018.

[53] W. Chen, I. Paik, and P. C. Hung, "Transformation-based streaming workflow allocation on geo-distributed datacenters for streaming big data processing," *IEEE Transactions on Services Computing*, 2016.

[54] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Transactions on parallel and distributed Systems*, vol. 27, no. 5, pp. 1344–1357, 2015.

[55] L. F. Bittencourt, E. R. Madeira, and N. L. Da Fonseca, "Scheduling in hybrid clouds," *IEEE Communications Magazine*, vol. 50, no. 9, pp. 42–47, 2012.

[56] H. Chen, J. Wen, W. Pedrycz, and G. Wu, "Big data processing workflows oriented real-time scheduling algorithm using task-duplication in geo-distributed clouds," *IEEE Transactions on Big Data*, 2018.

[57] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

[58] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 633–641.

[59] Y. Xia, D. Zhang, J. Kim, K. Nakayama, K. Zipser, and D. Whitney, "Predicting driver attention in critical situations," in *Asian conference on computer vision*. Springer, 2018, pp. 658–674.

Shihong Hu received the bachelor's degree in communication engineering from Jiangnan University in 2016. She is a PhD. candidate of the School of Artificial Intelligence and Computer, Jiangnan University. She had been a Visiting Scholar in Prof. Weisong Shi's MIST Lab for research on resource scheduling in edge computing project, Wayne State University, USA, from 2019 to 2020. Her research interests include wireless sensor networks and edge computing.

Guanghui Li received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2005. He is currently a Professor with the School of Artificial Intelligence and Computer, Jiangnan University, Wuxi, China. He has published over 70 papers in journal or conferences. His research interests include wireless sensor networks, fault tolerant computing, and nondestructive testing and evaluation. His research was supported by the National Foundation of China, Zhejiang, Jiangsu Provincial Science and Technology Foundation, and other governmental and industrial agencies..

Weisong Shi received the B.S. degree fromXidian University, Xi'an, China, in 1995, and thePh.D. degree from the Chinese Academy of Sci-ences, in 2000, both in computer engineering.Weisong Shi is a Charles H. Gershenson Distin-guished Faculty Fellow and a Professor of ComputerScience with Wayne State University, USA, wherehe directs the Mobile and Internet SysTems Labora-tory (MIST) and Connected and Autonomous dRiv-ing Laboratory (CAR), investigating performance,reliability, power- and energy-efficiency, trust andprivacy issues of networked computer systems, and applications. He is one ofthe world leaders in the edge computing research community and publishedthe first book on edge computing. His paper entitled "Edge Computing: Visionand Challenges" has been cited more than 1700 times. In 2018, Dr. Shiled the development of IEEE Course on Edge Computing. In 2019, Dr. Shiserved as the lead guest editor for the edge computing special issue on theprestigious Proceedings of the IEEE journal. He is the Founding SteeringCommittee Chair of the ACM/IEEE Symposium on Edge Computing (SEC)and the IEEE/ACM Connected Health: Applications, Systems and Engineering(CHASE). He is an IEEE Fellow and an ACM Distinguished Scientist.