A Blockchain-Based Containerized Edge Computing Platform for the Internet of Vehicles

Laizhong Cui, *Senior Member, IEEE*, Ziteng Chen, Shu Yang, Zhongxing Ming, *Member, IEEE*, Qi Li, *Senior Member, IEEE*, Yipeng Zhou, Shiping Chen, *Senior Member, IEEE*, and Qinghua Lu, *Senior Member, IEEE*

***Abstract*—Edge computing is promising to solve the latency issue in the Internet of Vehicles (IoV). However, due to decentralization, traditional edge computing suffers in management, deployment, and security. Containerization relaxes resource deployment and migration problems, but current container scheduling policies are inefficient to process complicated tasks based on directed acyclic graph or DAG structures. In this article, we design a containerized edge computing platform CUTE, which provides low-latency computation services for the Internet of Vehicles. The centralized controller is empowered with resource management and orchestration, and containers are scheduled to appropriate edge servers to optimize the computation delay. CUTE is also integrated with blockchain to improve network security. We formulate the vehicle task offloading and container scheduling problems and develop a heuristic container scheduling algorithm for DAG-based computation tasks submitted by vehicles remotely. We implement and deploy CUTE into the China Mobile Network, and conduct comprehensive experiments and a case study. The experiment results show that CUTE can provide low-latency computation services for vehicular applications and that the heuristic algorithm outperforms traditional container scheduling policies.**

***Index Terms*—Blockchain, container scheduling, edge computing, vehicle task offloading.**

Manuscript received March 30, 2020; revised July 15, 2020 and August 20, 2020; accepted September 20, 2020. Date of publication September 29, 2020; date of current version February 4, 2021. This work was supported in part by the National Key Research and Development Plan of China under Grant 2018YFB1800302 and Grant 2018YFB1800805; in part by the National Natural Science Foundation of China under Grant 61772345, Grant 61902258, Grant 61672358, and Grant 61836005; in part by the Major Fundamental Research Project in the Science and Technology Plan of Shenzhen under Grant JCYJ20190808142207420 and Grant GJHZ20190822095416463; and in part by the Pearl River Young Scholars funding of Shenzhen University. *(Corresponding author: Shu Yang.)*

Laizhong Cui is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: cuilz@szu.edu.cn).

Ziteng Chen, Shu Yang, and Zhongxing Ming are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: 1810272054@email.szu.edu.cn; yang.shu@szu.edu.cn; zming@szu.edu.cn).

Qi Li is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100876, China (e-mail: qli01@tsinghua.edu.cn).

Yipeng Zhou is with the Department of Computing, Faculty of Science and Engineering, Macquarie University, Sydney, NSW 2109, Australia (e-mail: ypzhou@mq.edu.au).

Shiping Chen and Qinghua Lu are with Data61, CSIRO, Sydney, NSW 1435, Australia (e-mail: shiping.chen@data61.csiro.au; qinghua.lu@data61.csiro.au).

Digital Object Identifier 10.1109/JIOT.2020.3027700

I. Introduction

RECENTLY, the Internet of Things (IoT) has been widely used in both daily consumer and industrial fields. Massive IoT devices are allowed to access the network to share resources, and cooperate as well as communicate with each other. According to [1], it is estimated there will be 75.44 billion IoT devices by 2025. As an important branch of IoT, the Internet of Vehicles (IoV) has become a growing research topic in recent years [2]. Empowered with computation and communication capabilities, smart vehicles improve driving safety and entertainment, and many innovative vehicular applications are developed as well [3], [4]. The massive vehicles will generate a huge amount of data and computation tasks, but IoV faces the challenge of how to deal with vehicular big data and tasks efficiently. Researchers proposed centralized cloud-based computing platforms to solve the problem, but they suffer from high network delay caused by long-distance data transmission from local vehicles to remote cloud centers [5]. Therefore, we should use other methods to improve the computation latency in IoV.

Edge computing is feasible to solve the latency issue in IoV [6], [7]. By deploying distributed servers at the edge end, vehicles can submit their tasks to the closer edge servers, which can process the vehicular big data and computation tasks. It avoids the long-distance data transmission to the remote computing centers such that the propagation latency can be greatly improved [8]. Nowadays, researchers have designed many innovative vehicular edge computing networks [9], [10]. However, it is challenging to manage the environments and configurations for distributed edge servers [11]. To satisfy various demands of applications, different software running environments need to be frequently migrated to edge servers, which is a time-consuming operation for network managers. Containerization provides the solution to the resource deployment and migration problems in edge computing [12], [13]. Source codes combined with relevant libraries and environments are encapsulated in containers, which can be easily deployed and migrated in different edge computing instances. Nevertheless, due to the lack of a global view, the resource management of conventional edge computing platforms remains difficult. For example, without a centralized scheduler, resource orchestration is inefficient, which will lead to network congestion, huge latency, and poor throughput [14].

In this article, we propose CUTE, a containerized edge computing platform for the Internet of Vehicles. It provides the containerized edge computing service for vehicles to offload their tasks remotely, bypassing the complicated configuration, migration, and deployment. We introduce the centralized controller to manage the distributed network and container orchestration. The controller will analyze the edge resource utilization and task information, then schedule the containers to appropriate edge servers. We formulate the vehicle task offloading and the container scheduling problems for CUTE to determine the task offloading strategy and optimize the computation delay for vehicles. On the one hand, CUTE has the advantages of edge computing, i.e., low latency, and containerization, i.e., easy deployment and migration. On the other hand, the CUTE controller can manage and migrate the distributed resources, as well as compute the appropriate container orchestration schemes according to the information.

In actual usage, CUTE faces several challenges. The first is network security. Distributed network entities can conduct misbehaviors by generating large amounts of junk data to attack the decentralized edge network [15]. With the development of Bitcoin [16] and Ethereum [17], blockchain provides a remarkable approach to improve the security for IoT networks [18], [19]. The reliable consensus mechanisms ensure that transactions in the blockchain are immutable, traceable, and auditable [20]. Therefore, we utilize the blockchain to improve the security of CUTE. Transactions generated by distributed parties will be bundled in blocks and appended to the blockchain. Moreover, network participants will periodically verify the transactions, and reward/punish the honest/dishonest parties. In this case, the security and reliability of CUTE can be guaranteed.

The second challenge is the container orchestration, as the container scheduling schemes directly affect the remote offloading delay for vehicles. Some container orchestrators are utilized in different computing platforms [21]–[23], but they employ straightforward container scheduling policies, such as first-in–first-out (or FIFO). For directed acyclic graph-based or DAG-based computation tasks, these simple scheduling policies cannot optimize the computation latency [24]. Therefore, we devise an efficient container scheduling policy for DAG-based tasks, which considers the data dependency, data size, computation overhead, and requirement of each module. The CUTE controller will run the scheduling algorithms to optimize the computation delay for vehicles.

We implement CUTE based on IBM Openwhisk and Ethereum and deploy CUTE into the China Mobile Network. We conduct a case study for CUTE and evaluate the container scheduling policy by generating different network conditions. The experiment results show that CUTE achieves the computation delay at the level of 100 ms, and the container scheduling policy outperforms the FIFO policy and distance first policy by 28.8% and 20.4%, respectively.

The main contributions are summarized as follows.

1) We develop a blockchain-based containerized edge computing platform CUTE for IoV. The centralized controller can manage the distributed edge servers and container orchestration. System transactions are recorded and verified in the blockchain to improve security.
2) We formulate the vehicle task offloading and container scheduling problems for CUTE and develop efficient container scheduling policy to optimize the offloading computation delay for vehicles.
3) We implement, deploy, and evaluate the performance of CUTE, and the experimental results show it can provide low-latency computation services for IoV.

The remainder of this article is organized as follows. Sections II and III will introduce the related works and the design of CUTE. Then, the problems and algorithms of vehicle task offloading and container scheduling will be discussed in Sections IV and V, respectively. The implementation, case study, and experiments for CUTE will be presented in Sections VI and VII. Finally, the conclusions will be given in Section VIII.

II. Related Work

A. Internet of Vehicles

IoV is a popular research topic in the academic and industrial communities [2]. Smart vehicles are empowered with computation, storage, and network such that they can communicate and corporate with others to improve driving safety and experience. Many applications are developed for smart vehicles, including virtual reality [3], social communication [25], autonomous driving [4], etc.

Nevertheless, current vehicular networks suffer the computation latency issue. Due to limited computing capability, vehicles cannot achieve satisfactory computation latency, especially when processing those complicated and latency-sensitive vehicular tasks [10]. Traditional cloud-based vehicular networks allow them to submit tasks to the high-performance cloud servers. However, the data transmission delay between vehicles and remote cloud servers is too high, which cannot meet the real-time demands of latency-sensitive applications [5]. Therefore, we will improve the vehicular network delay by utilizing edge computing technology.

B. Edge Computing

Edge computing improves network performances in terms of computation delay, bandwidth, etc. It has been utilized in the IoV field, which allows the smart vehicles to submit their tasks to the "closer" edge servers. Compared with traditional cloud-based vehicular networks, the computation delay of edge computing is improved, prompting the latency-sensitive vehicular applications to be integrated with edge computing for practical use [5]. Researchers also considered the vehicular task offloading problem in edge computing, in order to decide the vehicular tasks are offloaded locally or remotely to edge servers. For example, Du *et al.* [26] proposed a task offloading model to achieve the dual-side cost optimization of vehicles and edge servers. Qiao *et al.* [10] devised a collaborative task offloading model for multiaccess vehicular edge networks. Zhou *et al.* [27] optimized the energy consumption for vehicular edge computing systems.

Nevertheless, software compatibility remains challenging in edge computing [11]. Various applications are running in the edge computing platform, but corresponding software packages do not necessarily exist in the edge instances. Running

environments are frequently deployed and migrated to different edge servers, which will lead to an inefficient and complicated configuration. Meanwhile, due to decentralization, resource management and scheduling in distributed edge computing platforms are still difficult. Current edge resource scheduling relies on self-management of users or simple caching algorithms [28], but they cannot satisfy the delay requirements of massive IoT users. Additionally, traditional edge computing networks can be easily attacked by adverse parties due to the lack of authentication and traceability [15]. Distributed malicious users can conduct the Sybil attack to destroy the edge network by generating a huge amount of vehicles with fake locations and identifications [9]. These issues motivate us to develop an efficient and secure edge computing platform for IoV.

C. Containerization and Orchestration

Containerization settles the deployment and management hardships [29]. It provides multiple independent user-space instances at the operating system level and supports encapsulating the necessary source codes and software libraries. Containers can be run on top of the common operating system kernel and be migrated in different computing instances without complicated configuration and setups. In these years, containerization has been used in edge computing [12], [13], which is a feasible solution to resource deployment and management problems.

To orchestrate containers efficiently, researchers proposed several container orchestrators, such as Mesos [22], Borg [23], Kubernetes [21], etc. They manage the container amount, deployment, placement, scheduling policy, as well as the life cycle and health security, according to user settings and system conditions in terms of CPU, memory, and bandwidth [30]. Additionally, researchers also proposed other container scheduling schemes. For example, Chen *et al.* [31] proposed BIG-C, a preemptive container scheduling scheme to optimize the container computation latency. Based on Mesos, López-Huguet *et al.* [32] improved the Quality of Service (QoS) of the container orchestration given a specific deadline. However, current container scheduling policies are not efficient enough since they adopt straightforward but inefficient container scheduling policies such as FIFO [24]. When handling complicated DAG-based tasks, existing container scheduling policies cannot process the data dependency between a pair of modules. Meanwhile, they fail to consider the computation overhead and data size of tasks, as well as resource utilization of edge servers. Thus, we should improve the container scheduling policy for DAG-based tasks.

D. Blockchain Technology

Blockchain improves network security for edge computing [7]. It is a distributed database recording the transactions of all network participants to ensure immutability and consistency. Every node is allowed to verify the validity of the transactions. Blockchain provides secure mechanisms (e.g., Proof of Work and Proof of Stake) for decentralized edge nodes to achieve consensus without centralized authorities, which is suitable for distributed edge computing architecture.

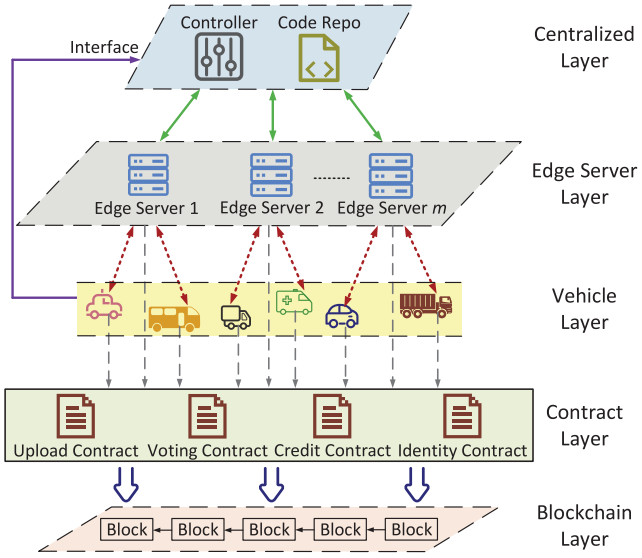


Fig. 1. Architecture of CUTE.

Nowadays, several blockchain-based edge computing platforms have been proposed. Sharma *et al.* [33] employed blockchain to synchronize and incentivize the distributed managers, each of which is empowered with attack detection and decentralized edge cluster management. Lei *et al.* [34] utilized blockchain to secure vehicle transportation in each edge domain. These works inspire us to build a blockchain-based edge computing platform to improve network security.

III. CUTE Design

A. Overview

In this article, we propose a CUTE computation platform for vehicles, which is depicted in Fig. 1. Vehicles in the network can offload their computation tasks to CUTE remotely or compute the tasks locally. When submitting to CUTE remotely, the computation tasks and corresponding containers will be scheduled to the proper edge servers to optimize the remote offloading computation delay for vehicles. The centralized controller will collect and analyze the task information generated by vehicles along with the resource utilization of edge servers, and compute the task offloading strategies and container scheduling schemes. Meanwhile, the transactions of distributed vehicles and edge servers will be bundled into blocks and appended to the tail of the chain structure. Network participants will verify blockchain transactions periodically by calling smart contracts.

CUTE adopts a five-layer hierarchical structure, which is divided into: 1) *centralized layer:* that comprises the controller for information analysis, system management, and scheduling, as well as the code repository for container image storage and retrieval; 2) *edge server layer:* that involves different edge computing instances to provide remote computation services; 3) *vehicle layer:* that consists of distributed vehicles, each of which will move at a certain speed and submit their tasks to CUTE or compute locally; 4) *contract layer:* that includes four smart contracts for the edge servers and vehicles to upload and

audit the blockchain transactions; and 5) *blockchain layer:* that records the transactions generated by distributed participants.

In CUTE, the computation processes are listed as follows.

1) Vehicles can apply for CUTE by sending requests to the interface provided by the centralized controller, which will determine the task offloading strategy for vehicles.
2) For the remote offloading strategy, the controller will compute the container scheduling scheme for DAG-based tasks according to the collected information. The scheme will be sent to edge servers and vehicles.
3) Target edge servers will download and deploy the container images from the code repository and bind with the vehicles directly.
4) The edge servers will receive the raw data, then return the computation results to the submitted vehicles.
5) The edge servers will call the upload contract to submit transactions to the blockchain layer, including the task information and computation results.
6) Distributed participants will periodically verify and vote for the transactions by calling voting contract, and reward/punish them by calling credit contract.

Assume a vehicle will move at a constant and finite speed, thus the network communication as well as the topology between vehicles and edge computing servers remain stable within the small fixed period. After receiving the computation results from CUTE, the moving vehicles can adjust their behaviors, e.g., accelerate or decelerate.

We also refer to serverless computing and the concept of Function as a Service (FaaS) [35]. We provide the unified interface and code repository in the centralized layer, consisting of basic container images and software packages, such as object detection, safety driving, virtual reality, etc. Vehicles can submit the task information, basic functions, and raw data to CUTE, bypassing the complicated environment and edge server configuration processes.

Generally, CUTE has the following benefits.

1) *Low Computation Delay:* By deploying edge servers at the edge end, vehicles can offload their computation tasks to the edge network remotely. Compared with traditional cloud computing, CUTE improves the computation latency such that vehicles can receive the results rapidly.
2) *Global Management:* Due to the global view of the controller, the decentralized edge network conditions and resource deployment can be analyzed and managed by the controller, making sure the system remains stable and efficient.
3) *Easy Deployment:* By utilizing containerization technology, vehicles can execute computation tasks or functions without configuration and resource allocation. Meanwhile, running environments can be rapidly migrated in different edge computing instances, which simplifies the resource migration and deployment.
4) *Efficient Scheduling Policy:* CUTE computes the task offloading strategies for vehicles. For the remote offloading, CUTE employs an efficient container scheduling policy to optimize the remote computation delay. The task offloading and container scheduling problems will be discussed in Section IV.
5) *Reliable Security:* Blockchain provides the mechanisms to improve the security of CUTE. Compared with traditional mechanisms where the centralized third parties manage the system security, blockchain eliminates unreliable authorities and maintains the overall security in a distributed manner. Decentralized participants share the ledger such that transactions cannot be tampered by misbehaving parties. Meanwhile, blockchain allows the decentralized peers to verify the transactions by voting, which improves the creditability and trustworthiness of CUTE further.

B. Controller

The CUTE controller is the core component of the system, which is responsible for managing the decentralized edge servers and vehicles, along with scheduling the computation resources and containers for vehicles. More specifically, the controller is in charge of the following.

1) *Information Collection:* It collects the network information (e.g., edge network traffic, vehicle locations, etc.), resource utilization of edge servers, and DAG-based task information.
2) *Network Management:* It manages the code repository and the distributed edge servers, including container deployment and orchestration, health monitoring, security and routing, etc.
3) *Strategy Decision:* It determines the offloading strategy and computes the container scheduling schemes to improve the computation delay for vehicles.

Meanwhile, the controller provides the unified interface for vehicles to access the edge computing resources, and also to calculate the payment a vehicle needs to pay for the task computation in CUTE.

C. Edge Servers

Edge servers are computing instances of CUTE, each of which is responsible for providing remote edge computing service and processing DAG-based tasks for vehicles. More specifically, after receiving the container scheduling schemes from the controller, corresponding edge servers will download and deploy container images from the code repository, then execute the tasks, and finally return the results to vehicles. The edge servers will bind with the vehicles and build a channel for direct communication to reduce the propagation latency. Besides, CUTE is scalable and flexible, as the edge servers can be dynamically added or removed by the controller according to demands or health conditions.

D. Decentralized Participants

Blockchain is adopted as the distributed database to record and verify the transactions of CUTE. The data structure of a transaction in CUTE is shown in Table I. A transaction consists of the task hash and relevant data hash, along with the identifications and signatures of submitted vehicles and edge servers,

TABLE I
DATA STRUCTURE OF A TRANSACTION

Task Hash	Data Hash	Vehicle Identity	Edge Server Identity	Vehicle Signature	Edge Server Signature	Time
0xb4a2d8a8...	0xe2afc5eb...	VE00000001	ES00000001	0xfdf3805c...	0xddf73e9c...	2020-3-28 16:42
0xa61d0abf...	0xb5e1739e...	VE00000003	ES00000005	0x3129e10e...	0x65c11d69...	2020-3-28 16:44
...	...	...	...	...	...	...

respectively. Meanwhile, the finish time of the task is included as well. In this article, we develop four smart contracts for the distributed participants to submit and audit transactions in the blockchain, including the following.

1) *Identity Contract:* It provides the basic CRUD functions to maintain the identifications of distributed edge servers and vehicles.
2) *Upload Contract:* It submits the transactions to the blockchain layer. Before uploading data to the blockchain, it will verify the identifications of edge servers and vehicles by calling the identity contract.
3) *Voting Contract:* It elects the consortium members and launches audits to vote for transactions. After voting, it will call the credit contract to reward/punish the honest/dishonest entities with blockchain tokens.
4) *Credit Contract:* It maintains the reputation for each decentralized participant.

We point out that the voting contract is the key blockchain design in CUTE. More specifically, the processes of voting contract are as follows. 1) in each round, a random number will be generated in a distributed manner, which will be used as the seed for member election. We refer to [36] and adopt the random number generation algorithm to guarantee randomness and security; 2) according to the random seed and reputation, every participant will generate a random number by running the verifiable random functions; and 3) the backbone protocol will sort the random numbers in a nonincreasing order and select r consortium members to vote for the validity of transactions.

After completing the computation tasks, the edge servers will collect necessary information and submit the transactions to the blockchain by calling the upload contract, which will verify the identifications by calling the identity contract, then upload corresponding data to the blockchain. In this case, the transactions with necessary information can be recorded in the distributed ledger and be verified by decentralized vehicles and edge servers.

The decentralized network entities will periodically and randomly verify the transactions. The backbone protocol will randomly elect the consortium members as we mentioned above. Within a consortium, the members will vote for transactions by calling the voting contract. If a transaction receives most votes (e.g., 2/3 majority votes), it will be considered valid. If the misbehavior of a dishonest entity is detected and considered invalid, it will be punished with blockchain tokens by calling the credit contract, and its reputation will decrease as well.

We note in traditional blockchain platforms, e.g., Bitcoin, the throughput is quite low, which is only seven transactions per second [20]. To improve the throughput of CUTE, we refer to [7] and adopt the Proof-of-Stake (or PoS) consensus mechanism based on reputation. The high-reputation distributed nodes are more likely to win the block mining opportunities and to append the blocks to the blockchain layer. In each round, CUTE will update the reputation of each node by calling the credit contract as well. In this case, we remove the time-consuming mining process of Bitcoin such that transactions can be appended to the blockchain layer rapidly, thus improving the throughput of CUTE. Meanwhile, the reputation-based PoS consensus mechanism can incentivize nodes to behave honestly and contribute their computing power to the transaction verification. Therefore, the mechanism is beneficial to the throughput and security of the blockchain.

At the same time, Cui *et al.* [7] proved that our scheme is resistant to the misbehaviors of distributed attackers. Hence, with the adoption of the blockchain, the security and trustworthiness of CUTE are guaranteed, and network participants will behave honestly and contribute to the system.

IV. PROBLEM STATEMENT FOR VEHICLE TASK OFFLOADING AND CONTAINER SCHEDULING

In this section, we will discuss the vehicle task offloading and container scheduling problems for CUTE. In this article, we will consider binary offloading, where vehicles are allowed to either compute the task locally or submit the task remotely to CUTE. If remotely, the corresponding containers will be scheduled to appropriate edge servers, and the latency performance of CUTE represents the remote offloading computation delay for vehicles. We will formulate the task offloading and container scheduling problems and optimize the vehicular offloading delay.

A. Notations

Suppose an edge server list $S = \{s_1, s_2, \ldots, s_m\}$, where $s \in S$ denotes a single edge server in CUTE. We assume there are vehicles $H = \{h_1, h_2, \ldots, h_n\}$, each of which is allowed to compute the task locally or submit the task to CUTE remotely. We denote $P(s)$ and $P(h)$ as the computing capability (i.e., CPU-cycle frequency) of edge server s and vehicle h, respectively. We also define $I(s)$ and $I(h)$ as the idle capacity of edge server s and vehicle h in terms of available memory for task computation. We assume a vehicle will move at a constant and limited speed, thus the network communication and data transmission between vehicles and CUTE are stable.

Given a specified duration, we assume there are computation tasks $T = \{t_1, t_2, \ldots, t_l\}$, each of which is generated by a certain vehicle. For a DAG-based task $t \in T$, it is described as a tuple $t = \langle G(t), \varphi(t), \sigma(t), \mu(t)\rangle$, where $\varphi(t)$ denotes

the required CPU cycles to finish the task, $\sigma(t)$ denotes the data size, and $\mu(t)$ denotes the computation requirement, i.e., required running memory. $G(t) = (V(t), E(t))$ represents the DAG structure of task t, where $V(t) = \{v_1, v_2, \ldots, v_p\}$ represents the modules of t, and $E(t) = \{e_1, e_2, \ldots, e_q\}$ denotes the data dependency of a pair of modules. For example, if modules v_i and v_j have data dependency, there exists an edge $e_{i,j} \in E(t)$ pointing from module v_i to v_j, and v_i must be finished before v_j. For a module $v \in V(t)$, we define prev(v) as the set of parent nodes, and succ(v) as the set of child nodes. prev(v) must be executed before module v, and succ(v) must be executed after module v.

For task $t \in T$, if it is offloaded remotely to CUTE, modules will be encapsulated into containers and then scheduled to the appropriate edge servers. We assume $C(t) = \{c_1, c_2, \ldots, c_k\}$ as the containers encapsulating the modules $V(t) = \{v_1, v_2, \ldots, v_p\}$. A container can encapsulate single or multiple modules simultaneously, and containers are also organized in DAG. For simplicity, we assume there is no data dependency between modules encapsulated in the same container. The values of required CPU cycles $\varphi(c)$, data size $\sigma(c)$, and requirement $\mu(c)$ of container c are the sum of encapsulated modules, and intercontainer data dependency also relies on the encapsulated modules [37].

Meanwhile, for each edge server $s_i \in S$, there is a container queue Q_i storing the containers queuing to be executed in s_i. We assume each edge server can handle only one container simultaneously. On the one hand, containers queuing in an edge server must be executed in a FIFO order. On the other hand, the head containers in different queues are required to be executed by the data dependency. If there is a data dependency between head containers in different queues, they cannot be executed in parallel.

B. Problem Definition

For the binary offloading model, vehicles can compute the tasks locally or submit them to CUTE remotely. We denote the local offloading delay of task $t \in T$ as $\mathcal{L}(t)$, and the remote offloading delay as $\mathcal{R}(t)$. We will discuss these two kinds of offloading delays as follows.

1) Local Offloading Delay: For the local offloading strategy, tasks generated by vehicles are executed locally without containerized computation. Generally, the local offloading delay $\mathcal{L}(t)$ relies on the required CPU-cycles of task t and the computing capability of vehicle h. Note that vehicles will compute the DAG-based tasks in a specific topological-sort order, and there is no data transmission overhead in local offloading. Therefore, for $t \in T$, if the idle capacity of vehicle h is no less than the memory requirement, i.e., $\mu(t) \leq I(h)$, the local offloading delay of task t generated by vehicle h is:

$$\mathcal{L}(t) = \frac{\varphi(t)}{P(h)}. \tag{1}$$

2) Remote Offloading Delay: When tasks are submitted to CUTE remotely, the modules will be encapsulated in containers. Generally, the remote offloading delay consists of: 1) *computation overhead:* the container execution period in edge servers, which is denoted as d_{comp} and 2) *transmission overhead:* raw data transmission period from vehicles to edge servers via the Internet, which is denoted as d_{comm}. We will discuss these two kinds of overheads, respectively.

Basically, for a DAG-based task, the computation overhead of remote offloading mainly relies on the container execution and queuing duration, as well as intermediate data communication overhead caused by data dependency. First, let us consider the intercontainer communication overhead of intermediate data. Suppose two containers, say c_a and c_b, are allocated to edge servers, say s_i and s_j, and there exists communication overhead $\omega(c_a, c_b)$ between c_a and c_b. According to [38], on the one hand, if they are scheduled to the same edge servers, i.e., $s_i = s_j$, the communication overhead $\omega(c_a, c_b) = 0$. On the other hand, if they are allocated to different edge servers, i.e., $s_i \neq s_j$, the communication overhead $\omega(c_a, c_b)$ relies on the data transfer size between the corresponding containers.

We define $\rho(c, s)$ as the *estimated start time* of container c being executed in edge server s, and define $FT(c)$ as the *actual finish time* of container c. $\rho(c, s)$ can be computed by

$$\rho(c, s) = \max\left(A(s), \max_{c_m \in \text{prev}(c)} (FT(c_m) + \omega(c_m, c))\right) \tag{2}$$

where $A(s)$ is the available time of edge server s to execute the container c, i.e., the container queue of s is empty. Intuitively, the start time of c should satisfy two conditions: 1) edge server s that executes c is available and 2) intermediate data from parent containers have been transferred.

We define $f(c, s)$ as the container allocating state function

$$f(c, s) = \begin{cases} 1, & c \text{ is allocated to } s \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

We call $M(s_i)$ as the finish time of edge server s_i finishing executing the containers in queue Q_i. It equals the finish time of the last container queuing in s_i

$$M(s_i) = \max_{c \in C(t)} \left\{ \left(\rho(c, s_i) + \frac{\varphi(c)}{P(s_i)} \right) f(c, s_i) \right\}. \tag{4}$$

Therefore, the overall computation overhead d_{comp} of task t is the maximum finish time of all edge servers, which can be computed by

$$d_{\text{comp}} = \max_{s_i \in S} M(s_i). \tag{5}$$

When offloading tasks to CUTE remotely, raw data must be transferred from vehicles to edge servers via the Internet. In this article, we only consider the communication delay of raw data transferred from the vehicle to the entry edge server that starts the computation process. We ignore the communication delay of results from the exit edge server to the vehicle, as the output data size is significantly smaller than the input raw data [39].

According to the Shannon–Hartley formula, the transmission rate $r(h, s)$ from vehicle h to edge server s is

$$r(h, s) = B \log_2 \left(1 + \frac{\theta \cdot \delta(h)}{\xi} \right) \tag{6}$$

where B and ξ mean the system bandwidth and noise power at the edge server, θ means the power gain, and $\delta(h)$ means

TABLE II
NOTATION LIST 1

Notation	Meaning
S	list of edge servers
s	an edge server
H	list of vehicles
h	a vehicle
$P(s), P(h)$	computing capability of s and h
$I(s), I(h)$	idle capacity of s and h
T	list of tasks
t	a computation task
$C(t)$	containers for t
c	a container
$G(t) = (V(t), E(t))$	DAG structure of task t
v	a module of t
$\varphi(t), \varphi(v), \varphi(c)$	required CPU cycles of t, v and c
$\sigma(t), \sigma(v), \sigma(c)$	data size of t, v and c
$\mu(t), \mu(v), \mu(c)$	requirement of t, v and c
$prev(v), prev(c)$	parent nodes of v and c
$succ(v), succ(c)$	child nodes of v and c

TABLE III
NOTATION LIST 2

Notation	Meaning
Q_i	container queue of edge server s_i
$\omega(c_a, c_b)$	transmission overhead of container c_a and c_b
$M(s)$	finish time of edge server s
$\rho(c, s)$	start time of c being executed in s
$FT(c)$	actual finish time of c
$A(s)$	available time of s
$f(c, s)$	container allocating state function
$r(h, p)$	data transmission rate from h to p
d_{comp}	computation overhead of remote offloading strategy
d_{comm}	transmission overhead of remote offloading strategy
α_t, β_t	offloading strategy state of t
$\mathcal{L}(t)$	local offloading delay of t
$\mathcal{R}(t)$	remote offloading delay of t
B	system bandwidth
θ	power gain
$\delta(h)$	transmission power of vehicle h
ξ	noise power

the transmission power of vehicle h. Thus, the raw data transmission overhead d_{comm} of task t can be computed by

$$d_{\text{comm}} = \frac{\sigma(t)}{r(h, s)}. \tag{7}$$

Consequently, the overall remote offloading delay $\mathcal{R}(t)$ of task t is

$$\mathcal{R}(t) = d_{\text{comp}} + d_{\text{comm}}. \tag{8}$$

C. Problem Formulation

We denote α_t and β_t as the offloading strategy state of t

$$\alpha_t = \begin{cases} 1, & \text{task } t \text{ is computed locally} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

$$\beta_t = \begin{cases} 1, & \text{task } t \text{ is computed remotely} \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

where $\alpha_t + \beta_t = 1$.

In this article, our objective is to minimize the maximum task computation delay, along with satisfying the capacity demand of edge servers and vehicles. Formally

$$\min \max_{t \in T} \ (\alpha_t \times \mathcal{L}(t) + \beta_t \times \mathcal{R}(t)) \tag{11}$$

$$\text{s.t.} \quad \begin{cases} I(h) \le \mu(t), & \alpha_t = 1 \\ \sum_{c \in C(t), s \in S} \mu(c) f(c, s) \le I(s), & \beta_t = 1 \\ \alpha_t + \beta_t = 1, & \alpha_t, \beta_t \in \{0, 1\}. \end{cases} \tag{12}$$

The notations are listed in Tables II and III.

V. VEHICLE TASK OFFLOADING AND CONTAINER SCHEDULING ALGORITHMS

In this section, we will study the container scheduling algorithm to optimize the remote offloading delay of CUTE and devise the task offloading strategy to minimize the overall computation delay for vehicles.

A. Optimal Container Scheduling

For the remote offloading strategy, we have the optimal container scheduling scheme, where containers are scheduled to the most appropriate edge servers, and the remote offloading delay is minimized. The optimal algorithm needs to consider all possible container scheduling schemes and selects the one with the lowest computation delay. We prove that the optimal container scheduling problem is NP-Hard.

Theorem 1: The optimal container scheduling problem is NP-hard.

Proof: It is easy to verify a given container scheduling scheme in polynomial time, thus the optimal container scheduling problem is in NP class. We prove the theorem by reduction from the maximum subset sum problem, which is NP-complete [40]. The maximum subset sum problem is, given a finite set U, for each $u \in U$, a value $z(u) \in Z^+$, a positive integer $F \in Z^+$, find a subset $U' \subseteq U$ such that $\sum_{u \in U'} z(u) \le F$ and $\sum_{u \in U'} z(u)$ is maximized.

Suppose two edge servers s_i and s_j, and the computing capability $P(s_i)$ is very large while $P(s_j)$ is very small. Therefore, to achieve a lower computation delay and reduce the communication overhead, the more containers scheduled to s_i, the better the latency performance will be.

Let U represent the containers to be scheduled, and $\{\mu(c) | c \in C\}$ is equal with $\{z(u) | u \in U\}$. Due to the huge computation deviation, s_i is the only edge server whose idle resource capacity $I(s_i) = F$, and the optimal scheduling scheme is to choose a set of task $C' \subseteq C$ such that $\sum_{c \in C'} \mu(c) \le I(s_i)$ and $\sum_{c \in C'} \mu(c)$ is maximized. Thus, the optimal container scheduling problem is equivalent to finding the maximum subset sum. ■

The optimal container scheduling scheme will be finished in nonpolynomial time, which cannot be used practically. When the numbers of containers and edge servers are large, the computation time for the optimal scheduling algorithm will explode exponentially. Therefore, to schedule the containers in polynomial time as well as achieve good latency performance at the same time, we refer to [41] and develop a heuristic container scheduling algorithm for DAG-based tasks.

B. Task Offloading Strategy With Heuristic Container Scheduling Algorithm

To reduce the time complexity of container scheduling in CUTE, for the remote offloading strategy, we develop a heuristic container scheduling algorithm for DAG-based computation tasks. We design a task offloading strategy based on a heuristic container scheduling algorithm, which will compare the local and remote offloading computation delays, and competitively select the offloading strategy for better latency performance.

1) Overview: The philosophies of task offloading strategy are: 1) computing the remote offloading delay and scheduling the containers to appropriate edge servers by the heuristic container scheduling algorithm and 2) selecting the task offloading strategy competitively to minimize the computation delay for each vehicle.

Generally, the task offloading strategy consists of two processes.

1) *Offloading Strategy Selection:* For each task generated by a specific vehicle, CUTE will evaluate the local offloading delay versus the remote offloading delay, respectively. Then, CUTE will competitively select the offloading strategy with lower latency.
2) *Container Scheduling Computation:* For the remote offloading strategy, the algorithm will compute the heuristic container scheduling scheme and remote offloading delay. Finally, it will send the remote computation latency to the offloading strategy selection process for decision.

In the container scheduling computation process, two steps are involved. 1) *module prioritization:* modules are prioritized according to data dependency of DAG structure and their computation information. Then, they are listed by their priorities and encapsulated in a container list in nonincreasing order; 2) *container list scheduling:* Containers in the list will be greedily scheduled to minimize the maximum finish time, according to the task and edge server information.

Before encapsulating in containers, we need to prioritize the modules of a task. We compute the priority $\text{pri}(v)$ of module $v \in V(t)$ by

$$\text{pri}(v) = \text{MET}(v) + \max_{v_m \in \text{succ}(v)} \text{pri}(v_m) \tag{13}$$

where $\text{MET}(v)$ is the minimal execution time of module v and $\text{MET}(v) = \min_{s \in S}([\varphi(v)]/[P(s)])$. Intuitively, modules that are closer to the entry and with lower required CPU-cycles will be ranked higher. Note that the entry and exit modules have the highest and lowest priorities, respectively. Consequently, modules can be ranked iteratively and encapsulated in nonincreasing order.

2) Details: We develop Algorithms 1 and 2, where Algorithm 1 represents the main function of the offloading strategy selection and Algorithm 2, also named as HCS, represents the subfunction for remote offloading delay computation for CUTE. From lines 3 to 6, Algorithm 1 will compute the local offloading delay. It will check whether the requirement of t is no more than the idle capacity of h. If satisfied, Algorithm 1 will compute the local delay $\mathcal{L}(t)$ by (1).

Algorithm 1: Offloading (T, S, H)

```
1  begin
2      for ∀t ∈ T do
3          if μ(t) ≤ I(h) then
4              Compute the local offloading delay L(t);
5          else
6              Let L(t) = ∞;
7          R(t) = HCS(t, S, h);
8          if L(t) ≤ R(t) then
9              Let t compute locally.
10         else
11             Let t compute remotely.
12         Update corresponding information for the next
           iteration.
```

In line 7, Algorithm 2 will be called for remote offloading computation. First, it will prioritize the modules of t by (13) and encapsulate modules in container list $C(t) = \{c_1, c_2, \ldots, c_k\}$. From lines 5 to 11, for each container c, it will consider each edge server. In line 4, we initialize an array TFA, where $TFA[i]$ implies the finish time when assigning c to s_i, and $TFA[i] = \rho(c, s_i) + ([\varphi(c)]/[P(s_i)])$. Then, in line 9, if the requirement is satisfied, HCS will compute the finish time $TFA[j]$. In line 11, HCS will evaluate the communication delay d_{comm} from vehicle h to the entry edge server by (7), and let $TFA[j] = TFA[j] + d_{\text{comm}}$. In line 12, HCS will allocate c_i to the edge server with the lowest delay, and let $FT(c_i) = \min TFA$. After considering the container list, HCS will return the maximum finish time meaning the remote offloading delay. Finally, Algorithm 1 will compare $\mathcal{L}(t)$ and $\mathcal{R}(t)$, then select the offloading strategy with lower delay, and finally update the corresponding information.

Theorem 2: The heuristic container scheduling algorithm has a competitive ratio of $(2 - [1/m])$.

Proof: We call HCS and *OPT* as the latency performance of heuristic and optimal container scheduling schemes, respectively. First, the optimal latency performance must be no less than the maximum finish time of containers and the average finish time of edge servers

$$\text{OPT} \geq \max_{c \in C(t)} FT(c) \tag{14}$$

$$\text{OPT} \geq \frac{1}{m} \sum_{c \in C(t)} FT(c). \tag{15}$$

Second, we suppose $c_i (c_i \in C(t))$ as the final container allocated to server s_j. Before allocating c_i to s_j, the finish time of s_j is not larger than the average finish time of other edge servers

$$M(s_j) - FT(c_i) \leq \frac{1}{m} \left(\sum_{c \in C(t)} FT(c) - FT(c_i) \right). \tag{16}$$

Algorithm 2: HCS (t, S, h)

```
begin
    Prioritize the modules in nonincreasing order, and
    encapsulate modules to container list
    C(t) = {c_1, c_2, ··· c_k};
    for i = 1 to k do
        Initialize TFA;
        for j = 1 to m do
            if I(s_j) ≤ μ(c_i) then
                Let TFA[j] = ∞.
            else
                Compute the finish time TFA[j] when
                assigning c_i to s_j.
                if i = 1 then
                    Compute the communication overhead
                    d_comm from h to s_j, and let
                    TFA[j] = TFA[j] + d_comm.
        Allocate c_i to the edge server with lowest finish
        time, and let FT(c_i) = min TFA.
        Update the idle resource of edge servers and
        vehicles.
    return max_{c∈C(t)} FT(c)
```

Hence, by rearranging (14)–(16), the latency performance of heuristic can be

$$\begin{aligned} \mathrm{HCS} &= M(s_j) \\ &\le \frac{1}{m}\left(\sum_{c\in C(t)} FT(c) - FT(c_i)\right) + FT(c_i) \\ &\le \frac{1}{m}\sum_{c\in C(t)} FT(c) + \left(1 - \frac{1}{m}FT(c_i)\right) \\ &\le \left(2 - \frac{1}{m}\right)\mathrm{OPT}. \end{aligned} \tag{17}$$

■

The CUTE controller will run Algorithms 1 and 2 to provide the task offloading strategy for vehicles. If offloading remotely to CUTE, vehicles can enjoy the containerized edge computing services with low computation latency.

VI. Implementation and Case Study for CUTE

In this section, we present the implementation of CUTE and conduct a case study based on object detection.

A. Implementation

We refer to [7] and implement the prototype of CUTE, which is shown in Fig. 2. CUTE is organized in a master-slave structure, where the centralized controller is allowed to collect and manage the distributed edge servers and vehicles. CUTE uses and improves IBM Openwhisk as the serverless computing platform, and Kubernetes as the container orchestration and management tool. We utilize Prometheus for container monitoring and log collection, and Grafana for data visualization. We also adopt RabbitMQ as the message broker to transfer messages, and Nginx as the Web server software to manage the load balancing, security, and traffic monitoring of CUTE. We employ Ethereum to record and verify the blockchain transactions. We develop the smart contracts in Solidity language and deploy them on Ethereum. We adopt and modify the PoS consensus mechanism to strengthen the throughput and security.

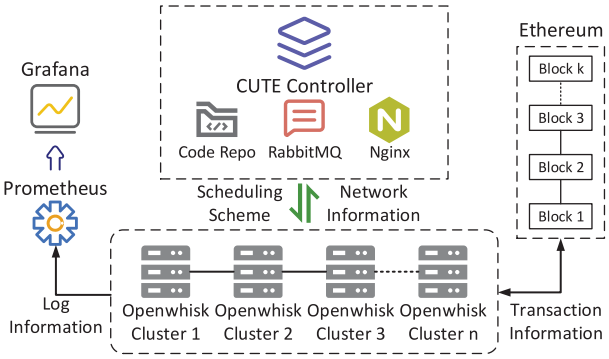


Fig. 2. CUTE implementation.

By sending URL requests to the interface provided by the controller, vehicles can access the edge resources rapidly and enjoy the high-performance containerized edge computing service without complicated deployment and management. Moreover, by utilizing Ethereum for the blockchain layer, transactions in CUTE are consistent, immutable, and verifiable such that the system security can be guaranteed.

B. Case Study

1) System Setup: We conduct a case study based on object detection, which is an important and widely used application in the IoV world. More specifically, distributed clients will submit videos or images consisting of various vehicle objects by sending URL requests to the interface. The controller will analyze the location and task information of clients and compute the container orchestration scheme. Target edge servers will download relevant Docker images from the code repository, then deploy the containers, and finally return the results to clients.

We deploy CUTE in the China Mobile Network, where the controller is placed in Hangzhou, China, along with two edge computing clusters in Hangzhou, China, and Shenzhen, China, respectively (as shown in Fig. 3). The clients are distributed in different locations and will submit a total of 60 object detection tasks. We will evaluate the latency performance of object detection, which is the remote offloading computation delay for vehicles. The configurations of the CUTE controller are listed in Table IV.

2) Results: The computation latency performance of different edge computing clusters is presented in Fig. 4, where the x-axis represents the index of tasks and the y-axis represents the latency performance. We observe that the Shenzhen edge cluster achieves 108.47 ms averagely, outperforming the Hangzhou edge cluster by 61.9%. This is because the distance from clients to the Shenzhen edge cluster is smaller

TABLE IV
CUTE CONTROLLER CONFIGURATION

Configuration	Parameter
CPU	4 core Intel Xeon (Cascade Lake) Platinum 8269C
Memory	8G
Operation System	Ubuntu 18.04
Network Bandwidth	100M

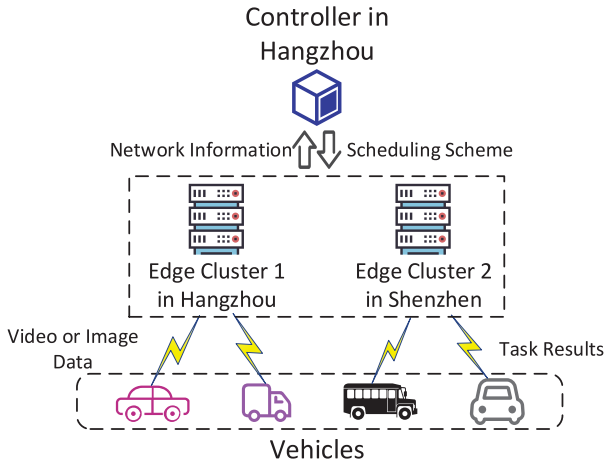


Fig. 3. Network topology.

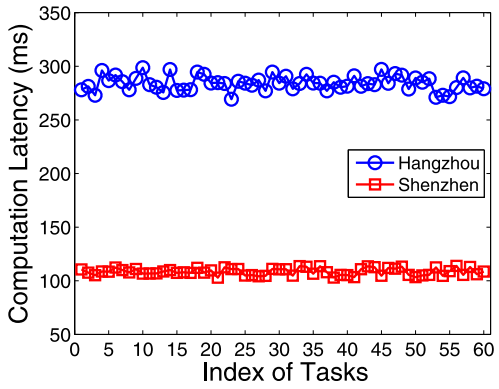


Fig. 4. Computation delay of tasks.

than the Hangzhou edge cluster, thus the data transmission latency dominates the latency performance. In this case, clients will offload their object detection tasks to the Shenzhen edge cluster remotely. The results show that clients can enjoy the containerized edge computing service at the level of 100 ms without complicated configuration and deployment.

Moreover, CPU utilization of the controller is shown in Fig. 5, where the x-axis and y-axis represent the index of tasks and utilization value, respectively. We see that the average controller's CPU utilization is 8.42%, and stays stable at a relatively low level. This means the CUTE controller can handle and schedule a large number of tasks, which is scalable and flexible.

VII. PERFORMANCE EVALUATION

A. Experimental Setups

We will evaluate the computation latency of local offloading and remote offloading strategies for vehicles. For comparison,

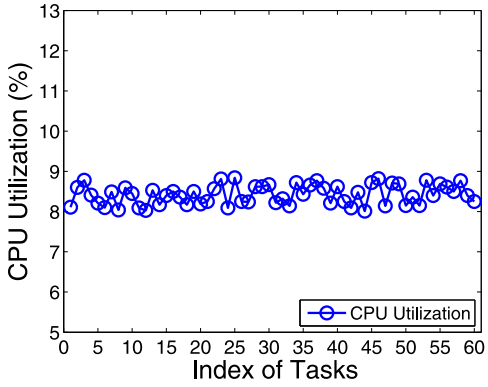


Fig. 5. CPU utilization of the controller.

we use the maximum delay ratio to measure the delay performance of different offloading strategies

$$\text{ratio} = \max_{t \in T} \frac{\mathcal{R}(t)}{\mathcal{L}(t)} \tag{18}$$

where ratio < 1 means the remote offloading strategy is better than the local offloading strategy in this situation, and *vice versa*.

For remote offloading strategy, put simply, we let each module be encapsulated in a single container. We will evaluate the container scheduling algorithm HCS, and compare it with traditional scheduling schemes. We develop the FIFO algorithm used in current container orchestrators, where containers are scheduled to edge servers in FIFO order. Additionally, we develop the distance first algorithm (or DF), a popular resource scheduling policy in edge computing, where resources are scheduled to the "closest" edge servers to minimize the data transmission distance [42]. If the "closest" edge server is overloaded and unable to satisfy the requirement demand, DF will proceed to search the second "closest" edge server, and so on. For remote offloading, we will evaluate and compare the performances of these three container scheduling policies.

We will generate different network configurations by changing the values of parameters, including the numbers of edge servers, vehicles, and containers per task, respectively. We will also adjust the overall data sizes of a task, as well as the computing capability of edge servers and vehicles. For simplicity, the required CPU-cycles of a task will grow linearly with the overall data size. Meanwhile, we will generate different vehicle distributions and traffic situations. More specifically, suppose the arrival rate of vehicles follows the Poisson process with parameter λ [43]. We will generate normal and heavy traffics by adjusting the value of λ, where the value of λ of heavy traffic is three times as the normal traffic. The default parameters are listed in Table V.

B. Experiment Results

1) Data Size: Figs. 6 and 7 show the relationship between the maximum delay ratio and data size. Generally, we can observe that as the data size grows, the maximum delay ratios of three algorithms will increase first, then decrease or stabilize. For example, in Fig. 6, when the data size grows from 8 to 128 MB, the ratio of HCS rises by 5.03%, and

TABLE V
DEFAULT VALUE OF PARAMETERS

Parameter	Default Value
CPU frequency of an edge server	16GHz
number of edge servers	3
CPU frequency of a vehicle	1.2GHz
number of vehicles	50
communication bandwidth	40MHz
power gain	-50dB
transmit power	23dBm
noise power	-114dBm
number of modules of a task	5
required CPU cycles of a module	0.2-0.3GHz
data size of a module	500-1500KB
Poisson parameter λ	500

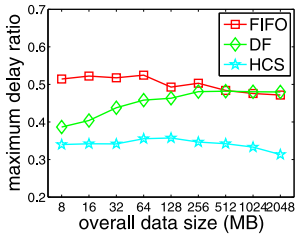


Fig. 6. Maximum delay ratio with different data sizes when $\lambda = 500$.

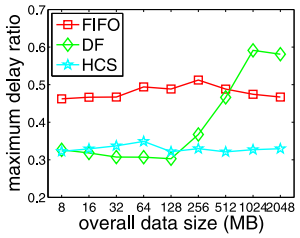


Fig. 7. Maximum delay ratio with different data sizes when $\lambda = 1500$.

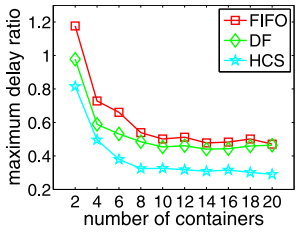


Fig. 8. Maximum delay ratio with different module numbers when $\lambda = 500$.

when the data size increases from 256 MB to 2 GB, the ratio grows by 11.4%. This is because, when the data size is relatively small, vehicles can still handle the simple tasks, and the communication delay dominates the latency performance of the remote offloading strategy. However, when the data size becomes large, the vehicles with limited capacity fail to handle the tasks efficiently, and the computation delay dominates the local offloading strategy. Similar trends can be observed in the heavy traffic situation in Fig. 7 as well. Moreover, when traffic flow becomes heavy, the performance of FIFO becomes unstable with larger fluctuations, and DF performs much worse with larger data size. For example, when the data size reaches 2 GB, the HCS outperforms DF by 43.3%. This is because the "closest" edge servers will be easily overloaded when data

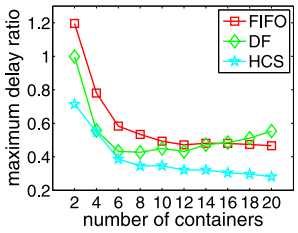


Fig. 9. Maximum delay ratio with different module numbers when $\lambda = 1500$.

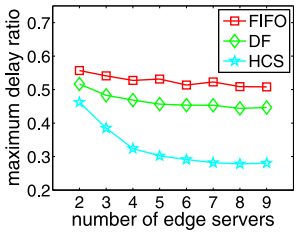


Fig. 10. Maximum delay ratio with different server numbers when $\lambda = 500$.

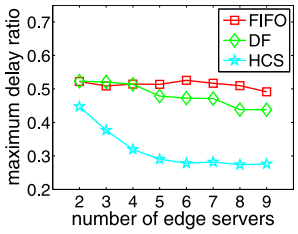


Fig. 11. Maximum delay ratio with different server numbers when $\lambda = 1500$.

size is large in heavy distribution. FIFO is unaware of network traffic and resource utilization, which will lead to inefficient container scheduling performance. The results also show that HCS works well in big data scenarios with different vehicle distributions.

2) Number of Containers: Figs. 8 and 9 present the relationship between the maximum delay ratio and the number of containers. As the number of containers rises, the remote offloading strategy is better than the local strategy since the computing capability of edge servers is much better than vehicles, and HCS always shows better performance than FIFO and DF. For example, in Fig. 8, when the number of containers increases from 10 to 20, HCS outperforms FIFO from 35.1% to 38.1%, and outperforms DF from 28.4% to 37.6%, proving that HCS works better with more containers. Meanwhile, when the traffic flow increases (Fig. 9), the performance gap between HCS and DF gets larger, where HCS performs 49.3% better than DF when the number of containers reaches 20. The reason is that the capacitated and "closer" edge servers are easily overloaded with more containers, meaning that DF cannot process complicated tasks with heavy traffics efficiently. Conversely, HCS can collect and analyze the resource utilization of edge servers, and compute the appropriate container scheduling scheme, which works well in different network situations.

3) Number of Edge Servers: We investigate the relationship between the maximum delay ratio and the number of edge servers, as depicted in Figs. 10 and 11. As the number of edge servers rises, the maximum ratios of the three algorithms

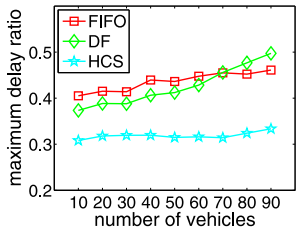


Fig. 12. Maximum delay ratio with different vehicle numbers when $\lambda = 500$.

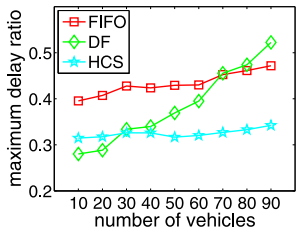


Fig. 13. Maximum delay ratio with different vehicle numbers when $\lambda = 1500$.

decrease first, then stabilize, especially for HCS. For example, in Fig. 10, when edge servers rise from 2 to 4, the delay performance of HCS improves by 30.2%, and remains at a low ratio as edge servers grow to 9. It means when the edge servers are insufficient, the delay performance of HCS deeply relies on the finite edge servers and becomes stable when edge servers are enough. Moreover, in heavy distribution (Fig. 11), the decreasing trends of FIFO and HCS are similar in general, while DF shows a stepped decline. For example, the delay performance of DF improves by 6.68% when edge servers grow from 4 to 5, and improves by 7.92% as they grow from 7 to 8, and remains stable at other times. The reason is that in heavy distribution, if the "latest" edge server is deployed closely, vehicles can submit their tasks with lower distance and the delay will decrease dramatically. But if it is deployed far away, vehicles will not submit tasks to the "far-away" edge server, which fails to contribute to the remote offloading delay.

4) Number of Vehicles: Figs. 12 and 13 present the relationship between the maximum delay ratio and the number of vehicles. In Fig. 12, as the vehicle number grows, the maximum delay ratios of FIFO and DF show the increasing trends while the ratio of HCS remains stable. For example, when the number of vehicles is only 10, HCS outperforms DF by 17.9% and FIFO by 24.7%, respectively. When the number of vehicles reaches 90, HCS outperforms DF and FIFO by 33.1% and 29.8%, respectively, proving that HCS works well with more vehicles. In Fig. 13, when the traffic is heavy, the delay of FIFO and HCS is similar to the normal traffic, while DF decreases a lot with vehicles growing. For example, when vehicles are only 10, HCS performs worse than DF, but when vehicles reach 90, HCS performs 35.6% better than DF. In heavy traffic, more vehicles access the closest edge server, but it can be overloaded with DF easily. Consequently, the containers need to be allocated to the far-away edge servers. As for HCS, it can collect the overall information and compute the appropriate allocation schemes so that the delay performance of HCS remains stable in heavy traffic.

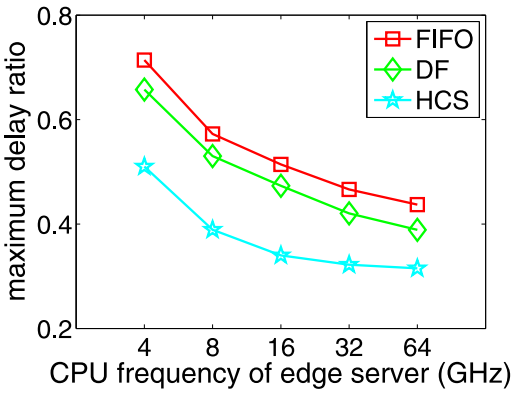


Fig. 14. Relationship between maximum delay ratio and edge server computing capability.

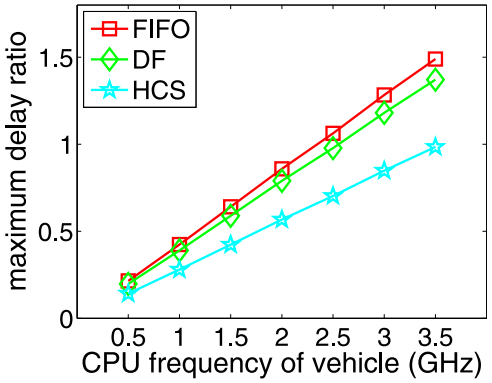


Fig. 15. Relationship between maximum delay ratio and vehicle computing capability.

5) Computing Capability of Edge Server and Vehicle: We also study the effects of computing capabilities on the computation delay, which are depicted in Figs. 14 and 15. In Fig. 14, when the computing capability of edge servers is only 4 GHz, the ratio gap between the three algorithms reaches the largest, where HCS outperforms FIFO and DF by 28.2% and 21.5%, respectively. It means HCS can achieve good performance even with limited computing capabilities. The ratio gap is getting smaller as the capability increases since the capability dominates the delay rather than algorithm superiority, but HCS still performs better than the other two. On the other hand, in Fig. 15, as the computing capability of vehicles grows, the maximum delay ratios increase linearly, meaning that the local offloading strategy is better with high-performance vehicles. For example, when the computing capability of vehicles is 3.5 GHz, the local offloading delay is smaller than the remote strategy. The results show that if computing capability is large enough, vehicles will offload their tasks locally. In the heavy traffic situations, the trends of these two figures do not change, meaning that the remote offloading delay with different computing capabilities is insensitive to vehicle topologies.

VIII. Conclusion

In this article, we proposed CUTE, a containerized and blockchain-based edge computing platform, to provide

low-latency computation service for IoV users. We introduced a centralized controller to collect system information and manage the distributed edge servers. Clients can submit computation tasks by accessing the unified interface. Corresponding containers are scheduled to appropriate edge servers to minimize the computation delay. CUTE also integrates with blockchain to improve the security. We formulated vehicle task offloading and container scheduling problems for CUTE and developed a heuristic container scheduling algorithm for DAG-based tasks. We conducted comprehensive experiments and a case study to evaluate the latency performances of CUTE. The results showed that CUTE is efficient for latency-sensitive vehicular applications, and our container scheduling policy outperforms the traditional algorithms.

REFERENCES

[1] J. Lee, D. Kim, H. Lee, Y. Lee, and J. H. Cheon, "Rlizard: Post-quantum key encapsulation mechanism for IoT devices," *IEEE Access*, vol. 7, pp. 2080–2091, 2019.

[2] B. Yin, Y. Wu, T. Hu, J. Dong, and Z. Jiang, "An efficient collaboration and incentive mechanism for Internet of vehicles (IoV) with secured information exchange based on blockchains," *IEEE Internet Things J.*, vol. 7, no. 3, pp. 1582–1593, Mar. 2020.

[3] L. Morra, F. Lamberti, F. G. Pratticó, S. L. Rosa, and P. Montuschi, "Building trust in autonomous vehicles: Role of virtual reality driving simulators in HMI design," *IEEE Trans. Veh. Technol.*, vol. 68, no. 10, pp. 9438–9450, Oct. 2019.

[4] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis, "A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 829–846, Apr. 2018.

[5] K. Xue, J. Hong, Y. Ma, D. S. L. Wei, P. Hong, and N. Yu, "Fog-aided verifiable privacy preserving access control for latency-sensitive data sharing in vehicular cloud computing," *IEEE Netw.*, vol. 32, no. 3, pp. 7–13, May/Jun. 2018.

[6] L. Cui *et al.*, "Joint optimization of energy consumption and latency in mobile edge computing for Internet of things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4791–4803, Jun. 2019.

[7] L. Cui, S. Yang, Z. Chen, Y. Pan, Z. Ming, and M. Xu, "A decentralized and trusted edge computing platform for Internet of things," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 3910–3922, May 2020.

[8] Z. Zhao, G. Min, W. Gao, Y. Wu, H. Duan, and Q. Ni, "Deploying edge computing nodes for large-scale IoT: A diversity aware approach," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3606–3614, Oct. 2018.

[9] S. A. Soleymani *et al.*, "A secure trust model based on fuzzy logic in vehicular ad hoc networks with fog computing," *IEEE Access*, vol. 5, pp. 15619–15629, 2017.

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

[11] 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 Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

[12] B. I. Ismail *et al.*, "Evaluation of docker as edge computing platform," in *Proc. IEEE Conf. Open Syst.*, Aug. 2015, pp. 130–135.

[13] G. A. Carella, M. Pauls, T. Magedanz, M. Cilloni, P. Bellavista, and L. Foschini, "Prototyping NFV-based multi-access edge computing in 5G ready networks with open baton," in *Proc. IEEE Conf. Netw. Softw.*, Jul. 2017, pp. 1–4.

[14] W. Yu *et al.*, "A survey on the edge computing for the Internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[15] 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 Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1508–1532, 2nd Quart., 2019.

[16] S. Nakamoto *et al.*, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.

[17] G. Wood *et al.*, *Ethereum: A Secure Decentralised Generalised Transaction Ledger* (Ethereum Project Yellow Paper), vol. 151, 2014, pp. 1–32.

[18] L. Cheng *et al.*, "SCTSC: A semicentralized traffic signal control mode with attribute-based blockchain in IoVs," *IEEE Trans. Comput. Soc. Syst.*, vol. 6, no. 6, pp. 1373–1385, Dec. 2019.

[19] M. Liu, F. R. Yu, Y. Teng, V. C. M. Leung, and M. Song, "Performance optimization for blockchain-enabled Industrial Internet of things (IIoT) systems: A deep reinforcement learning approach," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3559–3570, Jun. 2019.

[20] L. Cui, S. Yang, Z. Chen, Y. Pan, M. Xu, and K. Xu, "An efficient and compacted DAG-based blockchain protocol for industrial Internet of things," *IEEE Trans. Ind. Informat.*, vol. 16, no. 6, pp. 4134–4145, Jun. 2020.

[21] D. Bernstein, "Containers and cloud: From LXC to docker to kubernetes," *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, Sep. 2014.

[22] B. Hindman *et al.*, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proc. 8th USENIX Conf. Netw. Syst. Design Implementation*, 2011, pp. 295–308.

[23] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with borg," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, pp. 18:1–18:17.

[24] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. 13th EuroSys Conf.*, 2018, pp. 3:1–3:14.

[25] Y. Zhang, F. Tian, B. Song, and X. Du, "Social vehicle swarms: A novel perspective on socially aware vehicular communication architecture," *IEEE Wireless Commun.*, vol. 23, no. 4, pp. 82–89, Aug. 2016.

[26] J. Du, F. R. Yu, X. Chu, J. Feng, and G. Lu, "Computation offloading and resource allocation in vehicular networks based on dual-side cost minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1079–1092, Feb. 2019.

[27] Z. Zhou, P. Liu, Z. Chang, C. Xu, and Y. Zhang, "Energy-efficient workload offloading and power control in vehicular edge computing," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops*, Apr. 2018, pp. 191–196.

[28] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[29] J. Watada, A. Roy, R. Kadikar, H. Pham, and B. Xu, "Emerging trends, techniques and open issues of containerization: A review," *IEEE Access*, vol. 7, pp. 152443–152472, 2019.

[30] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and virtual machines at scale: A comparative study," in *Proc. 17th Int. Middleware Conf.*, 2016, pp. 1–13.

[31] W. Chen, X. Zhou, and J. Rao, "Preemptive and low latency datacenter scheduling via lightweight containers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 12, pp. 2749–2762, Dec. 2020.

[32] S. López-Huguet *et al.*, "A self-managed mesos cluster for data analytics with QoS guarantees," *Future Gener. Comput. Syst.*, vol. 96, pp. 449–461, Jul. 2019.

[33] P. K. Sharma, M.-Y. Chen, and J. H. Park, "A software defined fog node based distributed blockchain cloud architecture for IoT," *IEEE Access*, vol. 6, pp. 115–124, 2018.

[34] A. Lei, H. Cruickshank, Y. Cao, P. Asuquo, C. P. A. Ogah, and Z. Sun, "Blockchain-based dynamic key management for heterogeneous intelligent transportation systems," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 1832–1843, Dec. 2017.

[35] E. Jonas *et al.*, "Cloud programming simplified: A Berkeley view on serverless computing," 2019. [Online]. Available: http://arxiv.org/abs/1902.03383.

[36] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Oper. Syst. Principles*, 2017, pp. 51–68.

[37] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *J. Grid Comput.*, vol. 16, no. 1, pp. 113–135, 2018.

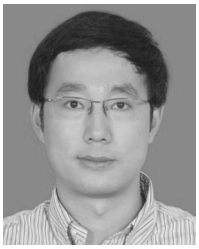
[38] L. Lv *et al.*, "Communication-aware container placement and reassignment in large-scale Internet data centers," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 540–555, Mar. 2019.

[39] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4924–4938, Aug. 2017.

[40] T. Sasamoto, T. Toyoizumi, and H. Nishimori, "Statistical mechanics of an NP-complete problem: Subset sum," *J. Phys. A, Math. Gen.*, vol. 34, no. 44, pp. 9555–9567, Oct. 2001.

[41] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

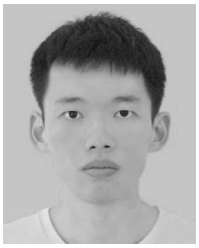
[42] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 38–43, Mar. 2017.
[43] S. Wang, T. Lei, L. Zhang, C.-H. Hsu, and F. Yang, "Offloading mobile data traffic for QoS-aware service provision in vehicular cyber-physical systems," *Future Gener. Comput. Syst.*, vol. 61, pp. 118–127, Aug. 2016.



Laizhong Cui (Senior Member, IEEE) received the B.S. degree from Jilin University, Changchun, China, in 2007, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2012.

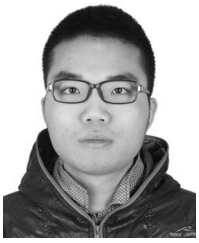
He is currently a Professor with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. He led more than ten scientific research projects, including the National Key Research and Development Plan of China, the National Natural Science Foundation of China, the Guangdong Natural Science Foundation of China, and the Shenzhen Basic Research Plan. He has published more than 70 papers, including IEEE TRANSACTIONS ON MULTIMEDIA, IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, *ACM Transactions on Internet Technology*, IEEE TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, and IEEE NETWORK. His research interests include future Internet architecture and protocols, edge computing, multimedia systems and applications, blockchain, Internet of Things, cloud and big data computing, software-defined network, social network, computational intelligence, and machine learning.

Prof. Cui serves as an Associate Editor or a Member of editorial board for several international journals, including *International Journal of Machine Learning and Cybernetics*, *International Journal of Bio-Inspired Computation*, *Ad-Hoc and Sensor Wireless Networks*, and *Journal of Central South University*. He is a Senior Member of CCF.



Ziteng Chen received the B.Sc. degree from Shenzhen University, Shenzhen, China, in 2018, where he is currently pursuing the M.Sc. degree.

His research interests include software-defined network, edge computing, and blockchain.



Shu Yang received the B.Sc. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2009, and the Ph.D. degree from Tsinghua University, Beijing, in 2014.

He is currently an Associate Researcher with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include network architecture, edge computing, and high-performance router.

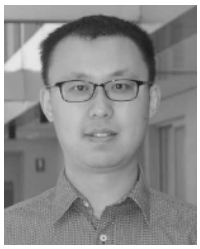
Zhongxing Ming (Member, IEEE) received the B.Eng. degree from the College of Software Engineering, Jilin University, Changchun, China, in 2009, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2015.

He is currently an Associate Research Fellow with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include future Internet architecture, Internet of things, blockchain, and edge computing.

Qi Li (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University, Beijing, China, in 2012.

He is currently an Associate Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. He has worked with ETH Zurich, Zürich, Switzerland, the University of Texas at San Antonio, San Antonio, TX, USA, the Chinese University of Hong Kong, Hong Kong, and the Chinese Academy of Sciences, Beijing. His research interests include network and system security, particularly in Internet and cloud security, mobile security, and big data security.

Dr. Li is currently an Editorial Board Member of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING and *ACM Digital Threats: Research and Practice*.



Yipeng Zhou received the M.Phil. and Ph.D. degrees from the Information Engineering Department, Chinese University of Hong Kong, Hong Kong, in 2008 and 2012, respectively.

From 2016 to 2018, he was a Research Fellow with the Institute for Telecommunications Research, University of South Australia, Adelaide, SA, Australia. From 2013 to 2016, he was a Lecturer with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. He is currently a Lecturer with the Department of Computing, Macquarie University, Sydney, NSW, Australia.

Dr. Zhou is a recipient of ARC DECRA in 2018.

Shiping Chen (Senior Member, IEEE) received the Ph.D. degree from the University of New South Wales, Sydney, NSW, Australia, in 2001.

He is a Principal Research Scientist with CSIRO Data61, Sydney. He is an Adjunct Associate Professor with the University of Sydney, Sydney, and the University of New South Wales, Sydney, through teaching and supervising Ph.D. students. He has been working on distributed systems for over 20 years with focus on performance, security, and trust. He is also actively involved in computing research community through publications, journal editorships, and conference TPC services, including WWW, EDOC, ICSOC, and IEEE ICWS/SCC/CLOUD. His current research interests include data services, secure data sharing, and services for collaboration. He has published more than 150 research papers in the above areas.

Qinghua Lu (Senior Member, IEEE) received the Ph.D. degree from the University of New South Wales, Sydney, NSW, Australia, in 2013.

She was a Researcher with the National ICT Australia, Sydney. She was an Associate Professor with the China University of Petroleum, Beijing, China. She is a Senior Research Scientist with Data 61, CSIRO, Sydney, as well as a Conjoint Senior Lecturer with the University of New South Wales. She has authored or coauthored more than 80 academic papers in international journals and conferences. Her current research interests include blockchain application design, blockchain as a service, self-sovereign identity, Internet of Things, reliability of cloud computing, and big data deployment.