Scalable Parallel Task Scheduling for Autonomous Driving Using Multi-Task Deep Reinforcement Learning

Qi Qi, Lingxin Zhang, Jingyu Wang, Haifeng Sun, Zirui Zhuang, Jianxin Liao, and F. Richard Yu, *Fellow, IEEE*

***Abstract*—The Internet of Vehicles (IoV) as a promising application of Internet of Things (IoT) has played a significant role in autonomous driving, by connecting intelligent vehicles. Autonomous driving needs to process the mass environmental sensing data in coordination with surrounding vehicles, and makes an accurate driving judgment accordingly. Since the vehicles always have limited computing resources, processing these data in parallel with efficient task scheduling is one of the most important topics. Most current work focuses on formulating special scenarios and service requirements as optimization problems. However, the complicated and dynamic environment of vehicular computing is hard to model, predict and control, making those previous methods unscalable and unable to reflect the real scenario. In this paper, a Multi-task Deep reinforcement learning approach for scalable parallel Task Scheduling (MDTS) is firstly devised. For avoiding the curse of dimensionality when coping with complex parallel computing environments and jobs with diverse properties, we extend the action selection in Deep Reinforcement Learning (DRL) to a multi-task decision, where the output branches of multi- task learning are fine-matched to parallel scheduling tasks. Child tasks of a job are accordingly assigned to distributed nodes without any human knowledge while the resource competition among parallel tasks is leveraged through shared neural network layers. Moreover, we design an appropriate reward function to optimize multiple metrics simultaneously, with emphasis on specific scenarios. Extensive experiments show that the MDTS significantly increases the overall reward compared with least- connection scheduling and particle swarm optimization algorithm from −16.71, −0.67 to 2.93, respectively.**

***Index Terms*—Parallel task scheduling, autonomous driving, deep reinforcement learning, multi-task learning.**

Manuscript received January 22, 2020; revised May 22, 2020 and September 26, 2020; accepted September 27, 2020. Date of publication October 9, 2020; date of current version November 12, 2020. This work was supported in part by the National Natural Science Foundation of China under Grants 62071067, 62001054, and 61771068, in part by the Beijing Municipal Natural Science Foundation under Grant 4182041, and in part by the Ministry of Education and China Mobile Joint Fund MCM20180101. The review of this article was coordinated by Dr. B. Mao. *(Corresponding authors: Jingyu Wang; F. Richard Yu)*

Qi Qi, Lingxin Zhang, Jingyu Wang, Haifeng Sun, Zirui Zhuang, and Jianxin Liao are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China, and also with the EBUPT.COM, Beijing 100191, China (e-mail: qiqi8266@bupt.edu.cn; zhanglingxin@ebupt.com; wangjingyu@bupt.edu.cn; hfsun@bupt.edu.cn; zhuangzirui@bupt.edu.cn; liaojx@bupt.edu.cn).

F. Richard Yu is with the Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario K1S 5B6, Canada (e-mail: RichardYu@cunet.carleton.ca).

Digital Object Identifier 10.1109/TVT.2020.3029864

I. Introduction

AUTONOMOUS Driving (AD), one of Intelligent Transportation System (ITS) pillars, is considered to have a bright future. Most AD functions, like self-navigation and collision avoidance, rely on the awareness of surrounding environments. For this reason, every autonomous vehicle is equipped with numerous cameras and sensors, to obtain environmental information. Moreover, autonomous vehicles also acquire relevant location and environment information observed by other surrounding vehicles, to ensure a timely driving decision, such as avoid traffic congestion and potential safety hazards. These miscellaneous data collected by various vehicles are arduous for a single vehicle to analyze and process.

Cloud data centers are highly computationally capable, but gathering all data into a cloud data center for centralized processing may result in a high communication cost. While for AD, a one-second delay in judgment can cause a serious traffic accident. Fortunately, along with the development of Internet of Things (IoT) and Vehicular Ad Hoc Network (VANET), the vehicles and the accessed edge computing nodes such as Base Stations (BSs) and Road Side Units (RSUs) endowed with server-level computing power construct a special mobile ad hoc network, which is referred to as the Internet of Vehicles (IoV) [1], [2]. The nodes in the IoV can contribute their computing resources for these complex real-time jobs. For example, vehicles within a VANET can issue requests to jointly complete data processing jobs with the assistance of the Multi-access Edge Computing (MEC) nodes and remote Data Centers (DCs) (as shown in Fig. 1). This manner of job assignment helps vehicles on the road or in parking lots make full use of on-board computing resources, providing an extensive range of consistent and reliable real-time services. Since the jobs for miscellaneous data analysis and processing are usually divided into multiple parallel child tasks, choosing an appropriate task scheduling scheme for a job becomes an important factor for the safe driving of autonomous vehicles.

Task scheduling has always been an indispensable and beneficial research topic especially for IoV systems with limited computing resources, and some progress has been made. Nevertheless, for the parallel task scheduling problem in IoV, there are still some challenges that have not been solved in previous work: (1) the vehicular environment is dynamic and

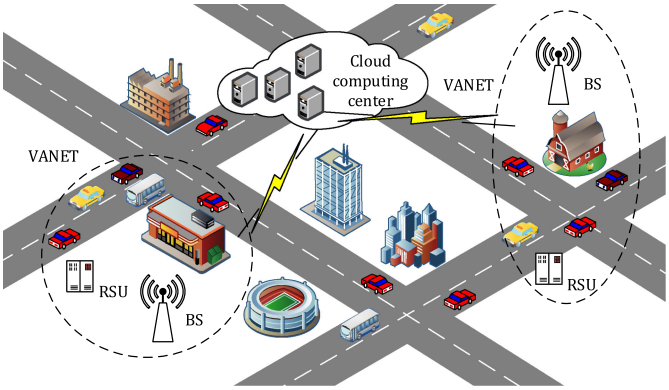

Fig. 1. Vehicles within a VANET can issue requests to jointly complete data processing jobs with the assistance of MEC nodes and remote DCs.

complicated, making it difficult to accurately model the dynamic change of vehicles or the specific way of processing tasks for heterogeneous computing nodes; (2) considering the resource competition between parallel tasks in one job, they need to be scheduled jointly, but the number of parallel child tasks may vary for different data processing jobs, which leads to a large discrete action space and an inconsistency of each task assignment decision; (3) for one request from a vehicle, its parallel data processing job requires to achieve the shortest response time, but for the whole AD environment, it is important to make the best use of computing resources of intelligent vehicles, edge computing nodes, and cloud data centers. We need to take into account various scheduling metrics (e.g., job completion time, total operation cost) and have a distinctive emphasis on different scenarios.

In a VANET, the computing power of each node and communication state between nodes all dynamically change, making some classic scheduling algorithms, e.g., Min-Min [3], Max-Min [4], and least-connection [5], not applicable in this scenario. Moreover, heuristic algorithms, such as Particle Swarm Optimization (PSO) [6], Genetic Algorithm (GA) [7], Ant Colony Optimization (ACO) [8], are also difficult to apply because they all require accurate modeling of the problem environment and then search approximate optimal solution based on the exact model. To overcome the first challenge, a model-free scheduling method based on Deep Reinforcement Learning (DRL) [9] is proposed to solve the task scheduling problem in IoV. DRL has the outstanding ability to handle complex control problems in high-dimensional state spaces and low-dimensional action spaces by utilizing deep neural networks [10]. Without any knowledge about the internal task execution mechanism of a VANET, the algorithm proposed in this paper can output the optimal task assignment scheme by learning the historical data and current network state, avoiding the procedure of accurate modeling.

For a data processing job during autonomous driving, multiple parallel child tasks need to share computing resources in a VANET and determine the final execution result of the entire job jointly. However, if the assignments of all child tasks need to be determined simultaneously, the size of action space will grow exponentially as the number of compute nodes and tasks increases. For example, when there are ten data acquisition nodes (the child task number is m) and they can be assigned to n available processors of various types in this VANET, the action space size of DRL is n^m. In previous work, these parallel tasks are almost scheduled independently a certain order. Although this method reduces the action space from n^m to n, it ignores the competition within child tasks, making it difficult to obtain the optimal allocation scheme.

To address the second challenge, a Multi-task Deep Reinforcement Learning approach for parallel Task Scheduling (MDTS) is proposed, which separates the output of the learned node allocation strategy in a multi-task learning manner for each task with considering the correlation among tasks. Multi-task learning is firstly introduced by Caruana in [11] to improve the learning of one task by exploiting the related information contained in other tasks. In multi-task deep reinforcement learning [12], agents are presented several related target tasks with shared characteristics. Rather than specialize on a single task, the objective is to generalize performance across all tasks. In this paper, a shared structure is employed to deal with multiple learning tasks jointly, which helps to improve the generalization of the model when appropriate tasks are chosen. With multi-task learning, the problem of inconsistent action space of different jobs is solved. When the DRL agent selects actions, it only needs to output the corresponding child task assignment schemes. As for neural network training, jobs are grouped according to their number of child tasks. When the historical data in a group are sufficient, the neural network parameters related to corresponding child tasks are trained. Eventually, the improved DRL-based scheduling algorithm can adapt well to the change of the number of tasks, and learn the relation among parallel tasks to achieve better scheduling.

To address the last challenge, we aim to optimize three metrics simultaneously: (1) minimizing the average job completion time to shorten response time for vehicle requests; (2)minimizing the average imbalance value of the VANET to improve the resources utilization ratio; (3) minimizing the total operation cost of all occupied computing nodes. A reasonable reward function is designed to optimize the performance of all three metrics at the same time, in which corresponding weight parameters are set for each metric. By adjusting those parameters in the reward function, we can easily implement various task scheduling requirements under different scenarios.

In summary, an improved model-free multi-task deep reinforcement learning algorithm is proposed in this paper to solve the scalable parallel task scheduling problem in IoV. The main contributions of this work include:

- A model-free scheduling method based on DRL is proposed to assign parallel tasks to nodes with different computing ability and resources (vehicles, edge computing nodes and cloud center server). This data-driven mechanism can learn to understand the workload and vehicular environment directly from experience, avoiding the accurate modeling of an intricate and dynamic environment.
- Selecting compute nodes for parallel tasks is abstracted as a multi-task learning problem, so that the resource

scheduling of autonomous driving is fine-matched to the multi-task learning with improved DRL architecture. Some fully-connected layers of the neural network are shared across tasks, followed by several task-specific output layers. This combination solves the problem of excessive action space of DRL and can adapt to various complex jobs with different numbers of child tasks. Furthermore, it is proved to be effective for improving task scheduling performance due to the consideration of correlation among parallel tasks.

- A reward function is designed to optimize the average job completion time, load imbalance value and total operation cost simultaneously. Through a simple adjustment of the weight parameters in reward function, personalized requirements in different scenarios of IoV can be completed.

The rest of this paper is organized as follows. Section II reviews related work about task scheduling methods relevant to this article. System model of parallel task scheduling in IoV is presented in Section III. In Section IV, we propose a multi-task DRL method for scalable parallel task scheduling. Section V discusses the evaluation results under extensive experiments and Section VI concludes this paper.

II. Related Work

In this section, we first review the scheduling algorithms, especially for parallel task scheduling, and then present some achievements and challenges about scheduling in IoV. The background about DRL is also provided in this section.

A. Task Scheduling and Resource Allocation Algorithms

A significant amount of research has focused on task scheduling and resource allocation [13]–[15]. Specifically, Zhang *et al.* [16] proposed a genetic algorithm to consolidate moldable VMs, reducing resource consumption. In [17], Zhu *et al.* devised a novel agent-based scheduling algorithm for real-time, independent and aperiodic tasks. Mao *et al.* [18] proposed a MapReduce task optimal scheduling algorithm, i.e. an improved MAX-MIN strategy, which can get the optimal MapReduce task resource allocation scheme. Wilczynski *et al.* [19] presented a model of the new cloud scheduler based on the blockchain technology, solving the scheduling optimization problem by using the asymmetric Stackelberg game model with the schedule execution time as privileged criterion. Nevertheless, most of the aforementioned algorithms need precise mathematical modeling and cannot address the large-scale dynamic scheduling issue efficiently, while it is apparently to be noted that in a vehicular cloud environment, tasks and resources are always dynamically varied and difficult to model.

Of course, some researches on parallel task scheduling problem in computing systems have been carried out. Sheikh *et al.* [20] proposed a task scheduling method based on a multi-objective evolutionary algorithm in order to determine the Pareto optimal solution while optimizing performance, energy, and temperature. In [21], an algorithm combining a double approximation scheme with a fast integer linear program was proposed, which determined the partitioning of tasks and the number of CPUs assigned to the moldable task. Pathan *et al.* [22] proposed a simple method of assigning a fixed priority to a subtask of a DAG job, and introduced a new technique for calculating the response time of each subtask based on the subtask priority. However, these parallel task scheduling algorithms are mostly based on dependent tasks, using DAG to settle parallel scheduling problems. Furthermore, in contrast to our expectation of an "intelligent" scheduling algorithm, these algorithms are complex and require modeling of the problem environment.

B. Task Scheduling and Resource Allocation for the Vehicular Environment

The vehicular environment consists of moving vehicles and stationary parked ones. The dynamic changes and instability of mobile vehicles bring great challenges to task scheduling and resource allocation for the vehicular cloud. The authors of [23] proposed the idea of Parked Vehicle Assistance, allowing parked vehicles to join VANETs as static computing nodes, and studied the corresponding task scheduling strategy. This method discovered neglected computing resources and helped deal with the poor connectivity of VANETs. Many other researchers have focused on the use of computing resources in moving vehicles. In [24], Wang *et al.* devised a model and an algorithm to describe and evaluate the serviceability of mobile vehicular cloudlets using a real-world large-scale urban vehicular mobility trace. Task replication policies were also designed to overcome the uncertainty of mobile vehicles and improve service reliability. However, this kind of task replication approaches can result in high resource costs. Actually, in addition to intelligent vehicles, computing nodes in VANETs also include various forms of edge computing nodes (e.g. BS, RSU) and remote cloud computing centers. The heterogeneity between these various computing nodes is also a problem need to be paid attention to when scheduling jobs in vehicular networks [25], [26].

So far, lots of task scheduling solutions, commonly referred to as task offloading [27]–[29], have been proposed in the vehicular environment. Liu *et al.* [30] expressed the multi-vehicle computational offloading problem in the vehicle edge network as a multi-user computational offloading problem and proposed a distributed computational offloading algorithm to calculate the equilibrium and reduce the delay of vehicle computational offloading. Huang *et al.* [31] proposed parking container edge computing (PVEC) based on container virtualization to perform tasks in containers deployed in PV with performance guaranteed, and studied the best task scheduling scheme from the perspective of maximizing social welfare. However, most of these solutions focused on establishing a specific system or framework with a rough scheduling algorithm rather than a concise system with an intelligent scheduling algorithm for vehicular environment.

C. Task Scheduling Algorithms Based on RL and DRL

The Reinforcement Learning (RL) solves the scheduling problems by maximizing the cumulative benefits of scheduling tasks and is proved to be efficient. Kontarinis *et al.* [32] focused

on the target of controlling leased resources and simulated the decision-making process of a controller implementing the Q-learning algorithm. Lin *et al.* [33] proposed an RL-based power management framework for data centers, which did not depend on the fixed assumptions of the task arrival and work service process. Yet, these papers have encountered great challenges in the design of state and action space. Because the convergence speed of traditional RL technology is usually proportional to the number of state-action pairs, while the complete resource allocation framework in computing systems always presents high dimensions in state and action space.

Deep learning has shown its strong capability of learning abstract representations from raw data [34]–[36]. The latest breakthrough in DRL in AlphaGo, combining the ability of decision-making with RL and the ability of perceiving with deep learning, and the development of Atari provides a good example of handling complex control problems. Many researchers also have tried to apply DRL to resource management and task scheduling problems. Mao *et al.* [37] built a system that learns to manage resources directly from experience with DRL, while it only considered the task scheduling for a single server rather than for a cluster with multiple compute nodes. Cheng *et al.* [38] proposed a novel DRL-based system with two-phase resource allocation and task scheduling to reduce energy costs for cloud service providers with large data centers and a large number of user requests with dependencies. Targeting at the problem of multi-objective work-flow scheduling in the heterogeneous IaaS cloud environment, Wang *et al.* [39] modeled the multi-objective workflow scheduling as a stochastic Markov game and develop a decentralized multi-task deep reinforcement learning framework that is capable of obtaining correlated equilibrium solutions of workflow scheduling. And in [40], DRL is used to solve the problem of offloading multiple service nodes and multiple dependencies of mobile tasks in a large-scale heterogeneous MEC.

Some researchers have begun to study how to apply DRL to solve the complex control problems in the Internet of vehicles. In [41], He *et al.* proposed an integrated framework in which the resource allocation strategy in vehicular network is formulated as a joint optimization problem. In this framework, deep reinforcement learning is used to realize the dynamic arrangement of networking, caching and computing resources to improve the performance of next generation vehicular networks. Ning *et al.*! [42] constructed a three-layer offloading framework in intelligent IoV to minimize the overall energy consumption while satisfying the delay constraint of users, and a deep reinforcement learning-based scheme was put forward to solve the optimization problem. In [43], a reinforcement-learning-based car following model for CAVs was proposed in order to obtain an appropriate driving behaviour to improve travel efficiency, fuel consumption and safety at signalized intersections in real-time.

In this paper, a multi-task decision-making method based on deep reinforcement learning is proposed, which aims to realize the reasonable allocation of computing tasks by learning the dynamic changing of computing nodes in a VANET.

III. Problem Formulation of Parallel Task Scheduling in IoV

In this section, we start with a system overview of the problem of parallel task scheduling in IoV and then describe the scheduling procedure in detail. Moreover, several optimization objectives are designed to comprehensively evaluate the scheduling effect.

A. System Overview

Several edge computing nodes and a group of vehicles with relatively low speed on the nearby road form a network through self-organization and connect with the remote cloud computing center. We solve the parallel task scheduling problem within each specific VANET. Considering a VANET consisting of N computing nodes (vehicles, other edge computing nodes and servers in cloud), each of them has different task processing capabilities, including CPU frequency and core count. All of these computing nodes are classified into three categories: $Requesters$, $Processors$, and $Controller$. $Requesters$ are the vehicles that generate jobs requiring collaborative processing, while $Processors$ are the vehicles or servers in cloud which are available for task processing. Generally speaking, the $Requesters$ are vehicles, and the $Processors$ can be vehicles, cloud servers, or other edge computing nodes. And the $Controller$, usually a designated edge computing node, receives job requests from $Requesters$ and assigns all related tasks to appropriate $Processors$. It is significant to note that the role of a vehicle in the network is not fixed to the $Requesters$ or $Processor$. Actually, a vehicle can even play these two roles at the same time. In addition, other edge computing node such as a BS can also act as both the $Controller$ and $Processor$. However, in a VANET, the $Controller$ is usually fixed and does not change during the scheduling process until the network dissolves.

In this work, the task assignment problem in VANET is abstracted as a parallel task scheduling problem. Job requests arrive at the $Controller$ continuously. Within one job, there are m child tasks that are executed independently. Assume that the number of child tasks m is variable but does not exceed the maximum value M, and the input data of each child task is stored on a certain vehicle. At the point when a job arrives, the $Controller$ simultaneously chooses corresponding execution nodes for each child task, giving full consideration to the resource competition among parallel tasks, so as to complete the job efficiently and speedily. Within a $Processor$, multiple tasks can be received at the same time and will be processed following FIFO rules. The complete system model is shown in Fig. 2.

Similar to the prior work [44], we assume that the information related to each job is known upon arrival; more specifically, the information mainly includes the number of child tasks and the details of each task (e.g., the length and storage location of pending data, number of CPU instructions). In this model, we assume no preemption but a fixed allocation profile, in the sense that the task won't be terminated from starting executing to completion. The notations are defined in Table I.

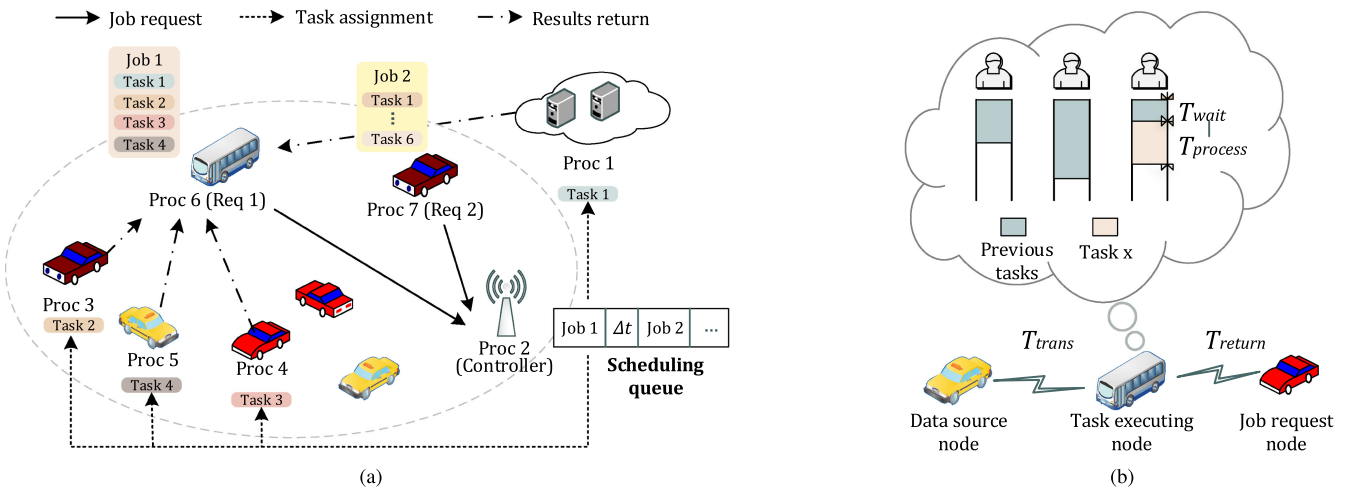

Fig. 2. System model of task scheduling in a VANET and task executing details. A vehicle called $Req1$ makes a job request consisting of 4 child tasks to the $Controller$, which then enters the scheduling queue and waits to be allocated. Then the $Controller$, an BS node, assign tasks to appropriate $Processors$ ($Proc1, 3, 4, 5$) according to the current state of the VANET. Each processor retrieves data from the data source node and performs the processing task. After execution, the results of child tasks are returned to $Req1$ for summary, and the job is completed.

B. Parallel Task Scheduling Procedure

In a VANET, vehicles constantly generate job requests, which are accepted by the $Controller$ and assigned to appropriate computing nodes upon arrival. We focus on the entire procedure of a job execution and model the problem for that process. According to reference [44], there are four stages for each job scheduling period:

Processor discovery: The $Controller$ detects adjacent $Processors$ throughout the VANET in which it is located, and selects the appropriate vehicles and edge computing nodes to form the set of $Processors$ to be selected together with the remote cloud servers. The driving status of all vehicles in the VANET, such as position, driving speed and moving direction, as well as the computing resources of all available computing nodes, can be observed by the $Controller$ through network communication protocol. At the scheduling decision time t, the processor set is denoted as $N(t)$. In this paper, the size of $N(t)$ is fixed as N and $N \neq 0$, indicating that only a part of nodes with strong computing power as $Processors$ rather than all the computing nodes in the network. At the beginning of each scheduling, all related information of processors in $N(t)$ will be updated.

Data transmission: After updating related status of all $Processors$, the $Controller$ schedules the current job, selects the appropriate execution location for each child task of the job, and then starts their execution process independently. The input data size of task $i(i \leqslant m)$ is expressed as $data_{i,t}$ (in bits), which needs to be sent from the original storage node to the task execution node.

It is assumed that the VANET includes Z_t communication channels denoted by $Z_t = \{1, 2, \ldots, Z\}$ at time t. For an upload communication link $z_{i,t}$ from source node $s_{i,t}$ to destination node $p_{i,t}$, the channel state is represented by $h^{(u)}_{z_{i,t}}$, the transmission power is $P_{z_{i,t}}$, and the interference power is $I^{(u)}_{z_{i,t}}$. The wireless channel state is assumed to be stable during each input data transmission. Given a fixed channel bandwidth W and noise power σ^2, the uplink transmission rate $r^{(u)}_{z_{i,t}}$ between $s_{i,t}$ and $p_{i,t}$ is [44], [45]:

$$r^{(u)}_{z_{i,t}} = W \log_2 \left(1 + \frac{P_{z_{i,t}} h^{(u)}_{z_{i,t}}}{\sigma^2 + I^{(u)}_{z_{i,t}}} \right). \tag{1}$$

Assuming that the frequency used by each channel is randomly assigned, the interference to one communication link mainly comes from other channels using the same frequency, which can be expressed as:

$$I^{(u)}_{z_{i,t}} = \sum_{k \in Z_t, k \neq z_{i,t}} \rho_k[z_{i,t}] P_k \tilde{g}_k, \tag{2}$$

where P_k denotes the transmission power of the k^{th} communication channel, $\tilde{g}_k$ is the interference power gain of the k^{th} communication channel, and $\rho_k[z_{i,t}]$ is the spectrum allocation indicator with $\rho_k[z_{i,t}] = 1$ if the k^{th} communication channel reuses the spectrum of channel $z_{i,t}$ and $\rho_k[z_{i,t}] = 0$ otherwise. And the data transmission delay $T_{trans}(i,t)$ is

$$T_{trans}(i,t) = \frac{data_{i,t}}{r^{(u)}_{z_{i,t}}} \tag{3}$$

While if the destination node is the remote cloud center, the vehicle first accesses to a BS which has a larger coverage and a mobility support. Then, the data are transmitted from the BS to the cloud center within the IP network hop-by-hop. The average uplink transmission between the BS and the cloud center is set to a constant R_{cloud}. Therefore, the data transmission delay $T_{trans}(i,t)$ is given by

$$T_{trans}(i,t) = \frac{data_{i,t}}{r^{(u)}_{z_{i,t}}} + \frac{data_{i,t}}{R_{cloud}} \tag{4}$$

Task execution: After the input data transmission is completed, the assigned tasks enter the queues of their respective $Processors$ for execution in accordance with FIFO rules. For

TABLE I
THE NOTATIONS DEFINATION

Notations	Defination
$data_{i,t}$	The input data size of task i at scheduling decision time t
$s_{i,t}$	The source node of task i at scheduling decision time t
$p_{i,t}$	The destination node of task i at scheduling decision time t
Z_t	All communication channels at scheduling decision time t
$z_{i,t}$	The communication channel from source node $s_{i,t}$ to destination node $p_{i,t}$
$h^{(u)}_{z_{i,t}}$	The uplink wireless channel state for $z_{i,t}$
$I^{(u)}_{z_{i,t}}$	The interference power for $z_{i,t}$
P_k	The transmission power of channel k
W	The channel bandwidth
σ^2	The noise power
$r^{(u)}_{z_{i,t}}$	The uplink transmission rate for $z_{i,t}$
$T_{trans}(i,t)$	The data transmission delay of task i at scheduling decision time t
R_{cloud}	The transmission rate between BS and the remote cloud centre
$\omega_{i,t}$	The computational intensity of task i at scheduling decision time t
$f_{n,t}$	The CPU frequency of $Processor\ n$ at scheduling decision time t
$T_{computation}(i,t)$	The computation delay of task i at scheduling decision time t
$T_{wait}(i,t)$	The time to wait in the compute node of task i at scheduling decision time t
$return_{i,t}$	The amount of data calculated by task i at scheduling decision time t
$T_{return}(i,t)$	The downlink transmission delay of task i at scheduling decision time t
η	The ratio of the output and input data
$T_{task}(i,t)$	The sum delay of processing child task i at scheduling decision time t
$T_{job,t}$	The job execution time at scheduling decision time t
T_{avg}	The average job completion time
$Load$	The load imbalance value
$TotalCost$	The total cost
$sumTime(p_i)$	The run time of node p_i
$avgTime$	The average running time of all N nodes
$Node_{cost}(i)$	The node cost for every time unit of node i

task i, the total workload is given by $data_{i,t}\omega_{i,t}$, where $\omega_{i,t}$ is the computational intensity (in CPU cycles per bit), indicating how many CPU cycles are required to process one bit of input data [46], which depends primarily on the nature of the application. The computing capability of $Processor\ n$ is described in terms of its CPU frequency $f_{n,t}$ (CPU cycles per bit), and it is assumed to be stable throughout the process of scheduling and execution each round. So the computation delay for each task is calculated by

$$T_{computation}(i,t) = \frac{data_{i,t}\omega_{i,t}}{f_{n,t}}, \tag{5}$$

and the whole task execution time is $T_{wait}(i,t) + T_{computation}(i,t)$ as shown in Fig. 2(b).

Results feedback: When task i completes, the selected $Processor\ p_{i,t}$ sends the result back to the $Requesters\ x$. Results return is still divided into two cases: vehicle or edge to vehicle node, and remote cloud computing centre to vehicle. It is assumed that the VANET includes $Z_t^{'}$ communication channels denoted by $Z_t^{'} = \{1^{'}, 2^{'}, \ldots, Z^{'}\}$ when task i completes. For a downlink wireless channel $z^{'}_{i,t}$ from $p_{i,t}$ to x, let $h^{(d)}_{z^{'}_{i,t}}$ represent the downlink wireless channel state and the interference power is denoted by $I^{(d)}_{z^{'}_{i,t}}$, which are assumed to be static during the transmission of each result. Thus, similar to Eq. 1, the downlink transfer rate from $Processor\ p_{i,t}$ to $Requesters\ x$ can be written as

$$r^{(d)}_{z^{'}_{i,t}} = W \log_2 \left(1 + \frac{P_{z^{'}_{i,t}} h^{(d)}_{z^{'}_{i,t}}}{\sigma^2 + I^{(d)}_{z^{'}_{i,t}}}\right). \tag{6}$$

The amount of data calculated by task i is represented by $return_{i,t}$ (in bits), so the downlink transmission delay from $Processor\ p_{i,t}$ to $Requesters\ x$ is

$$T_{return}(i,t) = \frac{return_{i,t}}{r^{(d)}_{z^{'}_{i,t}}}. \tag{7}$$

And when the execution node is cloud centre, the downlink transmission delay is

$$T_{return}(i,t) = \frac{return_{i,t}}{r^{(d)}_{z^{'}_{i,t}}} + \frac{return_{i,t}}{R_{cloud}}. \tag{8}$$

To simplify the calculation, we assume that the ratio of the output and input data $return_{i,t}/data_{i,t}$ is a constant η and remains unchanged in time, and $return_{i,t}$ can be represented as

$$return_{i,t} = \eta data_{i,t}. \tag{9}$$

Finally, the sum delay of processing child task i is given by

$$T_{task}(i,t) = T_{trans}(i,t) + T_{wait}(i,t) + T_{computation}(i,t) + T_{return}(i,t). \tag{10}$$

And the job execution time $T_{job,t}$ is the maximum $T_{task}(i,t)$ of child tasks within the group.

C. Scheduling Optimization Objectives

The average job completion time (T_{avg}) is set as the primary system objective. It's the most common and intuitive indicator of task scheduling optimization. The goal requires that the time span is minimized as much as possible in order to complete the job in a certain amount of time. In addition, most computing systems have clear requirements for load balancing so the reasonable resource utilization and network stability can be ensured. Therefore, the load imbalance value ($Load$) is set as an important indicator of the evaluation of scheduling algorithms in this paper. At the same time, the comparison of the total cost ($TotalCost$) is considered, with the goal of reducing the cost of task execution and the energy consumption of compute nodes.

They are calculated as:

$$Load = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}\left(sumTime\left(p_i\right) - avgTime\right)^2}, \tag{11}$$

$$TotalCost = \sum_{i=1}^{M} sumTime\left(p_i\right) \times Node_{cost}(i), \tag{12}$$

where $sumTime(p_i)$ means run time of node p_i, and $avgTime$ is the average running time of all N nodes; $Node_{cost}(i)$ is the node cost for every time unit. The smaller load imbalance value represents higher computing resource utilization rate and rationality of task assignment.

IV. A Multi-Task DRL for Scalable Parallel Task Scheduling: MDTS

In this section, we adopt DRL techniques to solve complex multiple task scheduling problems and describe how to represent the problem formulated in Section III as a DRL model. Importantly, we propose a multi-task DRL model for an appropriate scheduling algorithm, an improvement of A3C, and apply it to our problem model to realize the task scheduling of parallel tasks. Although DRL as a machine learning method cannot always provide 100% accuracy, the error only affects the efficiency of task execution for scheduling problems in AD. Therefore, DRL can be tried in resource management of vehicles [41] [42].

A. DRL Formulation

In a DRL model, an agent interacts with environment in discrete decision time steps. At each time step t, the agent obtains current state s_t (maybe high-dimensional) from the environment, which is perceived by a deep neural network to obtain a more specific and abstract state feature representation; then the agent takes an action a_t. Following the action, the environment returns the next state s_{t+1} and the reward r_t to the agent. Based on the obtained reward, the agent adjusts its decision strategy to achieve the objective that a state can be mapped to an action or to a probability distribution of actions with the highest discounted cumulative reward given by:

$$R_t = \sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t\right), \tag{13}$$

where $r(s_t, a_t)$ is the reward function and $\gamma \in [0, 1]$ represents the discount factor. Through the continuous loop of the above steps, the agent finally learns a stable optimal decision strategy.

The Asynchronous Advantage Actor-Critic (A3C) algorithm is an asynchronous deep reinforcement learning algorithm built on the top of the Actor-Critic method [47]. The A3C algorithm maintains a policy function $\pi(a_t|s_t;\theta))$, the actor, and an estimate of the value function $V(s_t;\theta_v)$, the critic. $\pi(a_t|s_t;\theta)$ represents the probability distribution of various action choices in the case of input state s_t. $V(s_t;\theta_v)$ is used to estimate the state value:

$$V\left(s_t;\theta_v\right) \approx \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \cdots | s_t\right]. \tag{14}$$

During each iteration, the actor selects a_t according to s_t and the policy function. Both of them are updated after every end of an episode (every $t_{\max}$ timestep or when a terminal state arrives). The loss of the full objective policy function takes the form:

$$L_{actor}\left(\theta'\right) = log\pi\left(a_t|s_t;\theta'\right) A\left(s_t, a_t;\theta,\theta_v\right). \tag{15}$$

$A(s_t, a_t;\theta,\theta_v)$ is an estimate of the advantage function given by:

$$A\left(s_t, a_t;\theta,\theta_v\right) = Q^{\pi}\left(s_t, a_t\right) - V\left(s_t;\theta_v\right). \tag{16}$$

For the A3C algorithm, n-steps returns are used to update the policy function and the value function, so the state-action value is calculated as:

$$\begin{aligned} Q^{\pi}\left(s_t, a_t\right) = {} & r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} \\ & + \gamma^n V\left(s_{t+n;\theta_v}\right). \end{aligned} \tag{17}$$

The entropy of the policy H is also added to the objective function to discourage premature convergence to suboptimal deterministic policies:

$$\begin{aligned} L_{actor}\left(\theta'\right) = {} & log\pi\left(a_t|s_t;\theta'\right)\left(R_t - V\left(s_t;\theta_v\right)\right) \\ & + \beta\nabla_{\theta'} H\left(\pi\left(s_t;\theta'\right)\right), \end{aligned} \tag{18}$$

where the strength of the entropy regularization term can be adjusted by β. The loss of value function is calculated as:

$$L_{critic}\left(\theta_v{}'\right) = \left(R - V\left(s_i;\theta_v{}'\right)\right)^2. \tag{19}$$

The A3C algorithm has multiple local agents executing in parallel, each of which learns and uploads its learning to a global agent. The global agent sends the learning to every local agent when it receives new learning. Since there is no correlation among multiple local agents and each of them experiences different states, the problem of correlation among training samples is well solved, making us free from the high consumption of experience replay (like DQN [9]). Some details can be seen in Fig. 3.

State Space: The state of the system, i.e., the specific situation of tasks waiting to be assigned and the current status of all computing nodes in the VANET, is set to an array. In order to fix the size of the neural network input of DRL, it is assumed that the number of child tasks does not exceed M.

We use an array with N elements to show the length and storage location of input data for each child task i, as $L_i = (l_1, l_2, l_3, \ldots, l_N)$. If the data is stored in node j, we set $l_j = data_i$ to data length and other elements to 0, and M arrays alike are used to show all the information of M tasks, as $L = (L_1, L_2, L_3, \ldots, L_M)$. Similarly, for all compute nodes, T is set to the estimated time for all tasks that are already in the queue of each child thread, and F_{node} represents the CPU frequency. Finally, all arrays above are connected into a new array, i.e. the system state information:

$$S = [L, T, F_{node}]. \tag{20}$$

In the case where the child tasks' number is less than M, all data of missing tasks are set to 0.

Action space: The system in this paper is set as an event-driven system, which means that the $Controller$ assigns tasks

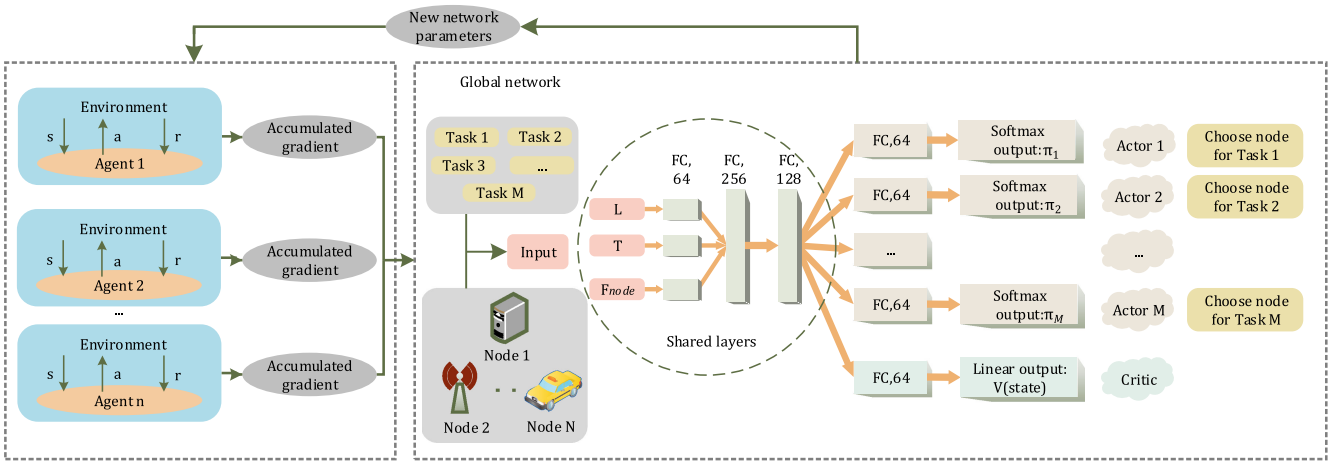

Fig. 3. Architecture of MDTS algorithm. For each MDTS agent, the current state of the VANET and the specific information of the job are taken as input and mapped to the action output through neural network, which is split into multiple branches corresponding to each child task.

upon arrival of a new job. Every task can be assigned to any of the N computing nodes, so a job with M child tasks would require a large action space of size N^M, which could make learning extremely challenging if we set the DRL output node directly to all possible policies. To solve this problem, multi-task learning is applied to DRL. The overall task assignment decision of one job is divided into M child decisions, corresponding to M child tasks. In this way, the final number of output nodes is reduced to $M \times N$. For each child decision, the action space is given by $\{1, 2, 3, \ldots, N\}$ where $a = i$ means "schedule the task to the i-th computing node". If the number of child tasks is less than M, the action of the corresponding output will be discarded directly. Finally, the action is given by

$$A = \{a_1, a_2, a_3, \cdots a_M\}, \tag{21}$$

where a_i is the server node that task i is assigned to; meanwhile, we set $a_i = \varnothing$ if task i does not exist.

Rewards: We craft the reward signal to guide the agent towards good solutions for our objective. In order to give consideration to multiple scheduling objectives and schedule tasks reasonably efficiently, we set the reward at each decision point to

$$r = \lambda \frac{T_{base} - T_{job}}{T_{base}} + \mu \frac{Load_{base} - Load}{Load_{base}} + \nu \frac{TotalCost_{base} - TotalCost}{TotalCost_{base}}, \tag{22}$$

where $T_{base}, Load_{base}, TotalCost_{base}$ are baselines of corresponding scheduling objectives which can be defined artificially according to the actual situation, and here $Load, TotalCost$ indicates internal situation of VANET after the completion of this scheduling. In addition, we can focus on one or two scheduling indexes by adjusting weight parameters λ, μ, ν. For example, when we want to minimize execution time without considering load balancing or cost, the parameters can be set as: $\lambda := 1, \mu := 0, \nu := 0$.

B. MDTS Approach

A deep neural network is used for the policy and value function. The input of the neural network is the state of a scheduling problem. As stated in section IV-A, we set up multiple actors responsible for individual scheduling of each child task. Accordingly, in this neural network, M softmax output branches are designed for policies $\pi_i(a_{t,i}|s_t; \theta_i')(i \in \{1, 2, \ldots, M\})$, and a liner output is designed for the value function $V(s_t; \theta_v')$, with several non-output layers shared. Each softmax output branch contains N output nodes, showing the probability distribution for assigning task i to these servers. This MDTS architecture is illustrated in Fig. 3. We propose a model with 3 fully-connected shared layers across child tasks followed by M task-specific actor branches and a critic output. For each task-specific branch, there is one fully-connected layer and an output layer. Parameters of shared layers can perceive resource contention among multiple child tasks, and learn to minimize the overall execution time of the whole job in each schedule. And parameters of task-specific actor branches can learn how to map the input information associated with each task to their output computing nodes respectively. While reducing the output nodes, this architecture guarantees the joint scheduling of multiple parallel child tasks by training sharing parameters.

Consequently, we change the loss function to:

$$L\left(\theta_1', \theta_2', \ldots, \theta_m', \theta_v'\right) = \frac{\sum_{i=1}^{m} L_{actor}\left(\theta_i'\right)}{m} + L_{critic}\left(\theta_v'\right), \tag{23}$$

where θ_i represents the parameters related to child actor network i. The proposed MDTS approach is presented in Algorithm 1. Different from the ordinary A3C, in which data (s_t, a_t, r_t, s_{t+1}) is used for all parameters in the deep neural network, a group of data is only used for training the parameters of critic network and child actor networks related to this job scheduling in the proposed MDTS. Compare with general multi-task learning, there is only one identical goal for all tasks in our system, arranging the most appropriate server allocation scheme for a

Algorithm 1: MDTS–Pseudocode for Each Learner Thread.

```
//Assume global shared parameters vectors
θ (θ_1, θ_2, ··· , θ_M) and θ_v and global shared counter
  C ← 0
//Assume thread specific parameter vectors
θ' (θ_1', θ_2', ··· , θ_M')and θ_v'
Initialize thread step counter t ← 1
repeat
    Reset gradients: dθ(dθ_1, dθ_2, ··· , dθ_M) ← 0
      ,dθ_v ← 0
    Synchronize thread-specific parameters θ' = θ and
      θ_v' = θ_v
    t_start = t
    Get task number m and state s_t
    repeat
        for i = 1 , i ≤ m do
            Select a_(t,i) according to policy
              π_i (a_{t,i}|s_t; θ_i')
        end
        Perform a_t = [a_{t,1}, a_{t,2}, ..., a_{t,m}]
        Receive reward r_t and new state s_{t+1}
        t ← t + 1
        C ← C + 1
    until terminal s_t or t − t_start == t_max;
    R = { 0                  for terminal s_t
        { V (s_t, θ_v')      for non − terminal s_t
    for i ∈ {t − 1, ··· , t_start} do
        R ← r_i + γR
        Accumulate gradients wrt θ':
          dθ ← dθ + ∇_θ' (Σ_{i=1}^{m} L_actor (θ_i')) /m
        Accumulate gradients wrt θ_v':
          dθ_v ← dθ_v + ∂L_critic (θ_v') /∂θ_v'
    end
    Perform asynchronous update of θ using dθ and θ_v
      of using dθ_v.
until C > C_max;
```

job consisting of several tasks so that the duration of the job execution can be the shortened as much as possible. Therefore, there is no specific reward for the output of each child actor, and r is used to train the parameters of all actor networks.

Our proposed architecture realizes the scalability of tasks through the special design of state space, action space and neural networks training. For state space, if the number of child tasks is less than M, all information like input data length of missing tasks are set to 0. While for action space, If the number of child tasks is less than M, the action of corresponding output will be discarded directly. When we train all neural networks, a group of data is only used for training the parameters of critic network and child actor networks related to this job scheduling. The advantage of this architecture is that it can schedule all jobs with different number of tasks. However, the disadvantage is also obvious, that we must keep the number of tasks in each job does not exceed M. For neural networks, the scalability of input data has always been a difficult problem, and we will continue our study and seek a better solution for scalable parallel task scheduling.

When evaluating RL algorithms, there are some essential traits to consider like computational complexity and sample complexity [48]. At first, the proposed MDTS algorithm exhibits low online computational complexity like [49], i.e., the computational complexity is proportional to the number of actions (number of compute nodes multiplied by the number of child tasks) at each decision epoch, which is insignificant for cloud computing systems. Sample complexity measures the amount of time-steps for which the algorithm does not behave near optimally or, in other words, the amount of experience it takes to learn to behave well, which is the metric that we usually focus on the most. And for this work, we improve DRL algorithm with multi-task learning for parallel task scheduling, which has been proved helpful in reducing sample complexity [50].

V. Evaluation

In this section, we conduct extensive simulations to evaluate the proposed MDTS algorithm applied to the problem model designed in Section III. These experiments are simulated through python, and the neural networks involved are implemented by TensorFlow. All the experiments are completed by using merely a normal desktop with an Intel Core 2.6 GHz CPU with 8 GB memory because of the low calculation requirement of the model and algorithm.

We assume that there are 8 vehicular nodes with one other edge computing nodes available in the VANET we consider, and jobs with no more than 10 child tasks arrive at regular intervals ($M = 10, N = 10$). Supposing that the relative distance between vehicles in the VANET remains stable during the time we discuss, and the number of CPU cores, the basic values of CPU frequency (GHz), and the cost for every unit time of all computing nodes are given in Table II. In this paper, the number of CPU cores is reflected as task number that the server can process synchronously, or the number of child threads that the server can provide for processing tasks. The basic value of input data for each child task are $[51, 37, 61, 86, 15, 71, 37, 61, 116, 35]Mbits$. Both of actual CPU frequency values and input data follow normal distribution with those basic values as mean, and the computational intensity of all task is set to $\omega_0 = 1000$ Cycles/bit.

The wireless channel state is modeled by an inverse power law $h^{(u)}_{z_{i,t}} = h^{(d)}_{z'_{i,t}} = A_0\, l^{-2}$, where A_0 = -17.8 dB and l represent the distance between the source node and the destination node. And $\tilde{g}_k = B_0\, l^{-2}$, where B_0 = -57.8 dB and l represent the distance between the source node of interference channel and the destination node of the actual concerned channel. It is assumed that the transmission power of all channels is $P = 0.1$ W, and other default parameters in vehicular system are: transmission constant $R_{cloud} = 100$ Mbps, channel bandwidth $W = 10$ MHz, and noise power $\sigma^2 = 10^{-13}W$ [44]. Furthermore, we divide the spectrum resources into 10 blocks, and assume that the all data transmission channels of tasks for one job chooses

TABLE II
THE NUMBER OF CPU CORES AND THE BASIC VALUE OF CPU FREQUENCY (GHZ) OF COMPUTING NODES

	node1	node2	node3	node4	node5	node6	node7	node8	node9	node10
CPU cores	4	3	1	1	3	4	1	3	3	2
CPU frequency	12.6	8.0	3.6	4.1	2.4	3.1	2.8	4.4	6.5	3.0
$Node_{cost}$	8	4	2	2	2	2	2	2	2	2

different frequency blocks. The frequency used by each channel is randomly assigned.

Besides, the average theoretical job execution time in this model is set to 125% of the average job arrival interval ($TaskDemand = 125\%$). Meanwhile, we experiment with other $TaskDemand$ conditions vary between 60% to 160% as well. Furthermore, some parameters in the algorithm are set as follows: $\gamma := 0.9$, $\beta := 0.001$, $t_{\max} := 20$, and learning rate of networks $\alpha := 0.00005$. During the algorithm training process, there are 4 local agents to achieve asynchronous learning. As for the reward function, in order to consider multiple scheduling indicators simultaneously and focus on shortening the average job completion time, the parameters are set to: $\lambda := 6, \mu := 1, \nu := 3$, and the baselines of corresponding scheduling objectives are: $T_{base} = 12, Load_{base} = 5, TotalCost_{base} = 80$. Under the above assumptions, there is almost no interference because the job arrival interval and data processing time are much longer than data transmission time.

We compare our MDTS algorithm with two basic task scheduling algorithms:

- **Least-Connection Scheduling (LC):** The LC scheduling algorithm directs pending task to the server with the least number of established connections.
- **Particle Swarm Optimization (PSO):** PSO optimizes a scheduling problem through iterative attempt to improve a candidate solution with regard to a given measure of quality.

Actually, in terms of algorithm design, MDTS has shown its superiority since it does not require fine-grained modeling of problem environments and manual extraction of environment features, which are necessary for those baseline methods. Detailed comparison and analysis of those basic algorithms prove that using MDTS approach to solve this parallel task scheduling problem can practically get better performance.

In the first place, the convergence of proposed MDTS are verified. Fig. 5 shows the average reward of the three scheduling algorithms over 1000 iterations. Obviously, as the number of iterations increases, MDTS manifests a better scheduling effect with more stable and higher reward while other two scheduling algorithms show strong oscillations. We make detailed comparisons based on T_{job}, $Load$, $TotalCost$ (defined in section III) of 100 task scheduling in a stable phase for each algorithm, and $ScheduleTime$, which represents the time it takes for each algorithm to complete 25000 times of scheduling. As shown in Table III, MDTS not only outperforms LC and PSO in terms of the overall reward index, but also significantly reduces the job execution time by 60.3% and 33.4% respectively, with still the lowest total cost. As far as this figure is concerned, although

TABLE III
AVERAGE INDICATOR VALUES IN STABLE PHASE

	Time	Load	Total Cost	Reward	Schedule Time
LC	20.90	9.21	181.53	-16.71	10.39
PSO	12.46	5.26	83.48	-0.67	1120.89
MDTS	8.30	1.41	70.37	2.93	406.31

TABLE IV
AVERAGE INDICATOR VALUES IN STABLE PHASE WITH INTERFERENCE

	Time	Load	Total Cost	Reward
LC	25.22	9.821	197.52	-11.98
PSO	15.71	5.58	95.19	-2.54
MDTS	12.32	2.16	80.07	0.40

PSO can achieve scheduling effect most close to MDTS, it needs a high time cost to complete the scheduling as a search-based algorithm, more than 2 times that of MDTS. To prove our conclusion more rigorously, we adjusted the bandwidth to $W = 1$ MHz to increase data transmission time, and observed the performance of each algorithm in the presence of interference, which is shown in Table IV. These experimental data indicate that MDTS, without knowing anything about task execution rules within computing nodes, can formulate sensible task assignment plans based only on experience and the current state of network environment.

More detailed data concerning the above experiment is presented in Fig. 4. In this experiment, the number of parallel tasks is variable, so we compare the average indicators for jobs with different numbers of child tasks. As a search-based heuristic algorithm, PSO obtains optimal scheduling performance for jobs with a small number of tasks. However, as the number of tasks increases, the search space of PSO grows exponentially. At this time, MDTS displays its advantages, getting the highest reward when the number of child tasks is greater than 3. For other individual scheduling metrics, compared with PSO, MDTS achieves an average reduction of 41.78%, 78.91% and 30.35% on job completion time, load imbalance value and total cost respectively when the number of child tasks is 8. It's the main reason why MDTS performs best when discussing the average scheduling metrics of all jobs. It considers the resource competition among multiple jobs because the maximization of the decision process is the cumulative reward. Specifically, MDTS may reserve a certain amount of excellent computing resources for a job with a large number of tasks. Though this policy may result in a suboptimal completion time for an uncomplicated job,

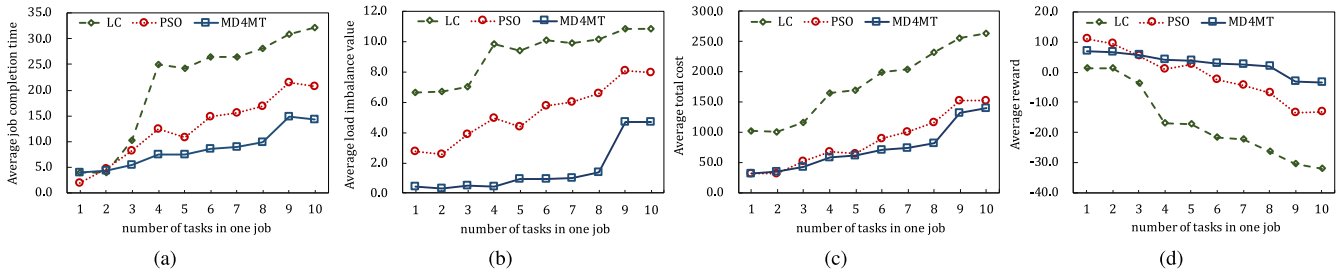

Fig. 4. Average job completion time (a), load imbalance value (b), total cost (c), and overall reward (d) for jobs with different number of child tasks.

TABLE V
THE PROBABILITY DISTRIBUTION OF EACH SCHEDULER ASSIGNING TASKS TO SERVER NODES

	node1	node2	node3	node4	node5	node6	node7	node8	node9	node10
MDTS	72.446%	9.631%	0.683%	1.396%	0.428%	0.449%	0.379%	0.284%	13.302%	1.001%
PSO	4.515 %	25.037%	11.623 %	6.306 %	5.056 %	10.019%	3.134 %	12.724 %	18.825%	2.761%
LC	23.872%	20.985%	13.825 %	6.530%	5.570%	5.918%	5.698 %	6.497%	6.977 %	4.130%

it can ensure that the overall average job completion time and other indicators are minimized.

To illustrate the intelligence of MDTS more clearly, we calculate the probability distribution of each scheduler assigning tasks to these server nodes in the task scheduling stable phase. Table V gives these probability distributions. Compared with other schedulers, MDTS is aware of the high $TaskDemand$ and relies on the remote cloud node accordingly, to complete jobs more efficiently through its powerful computing capability. Besides, it can be also observed that when consider of vehicular and other edge computing nodes, MDTS prefers to choose nodes with higher computing power and lower cost. For instance, the probability of selecting node2 and node 9 for MDTS is 9.631% and 13.302% respectively, much higher than other nodes. More surprisingly, MDTS perfectly balances load imbalance value with average job execution time and total cost. As shown in Table III and Fig. 4, it maintains the lowest load imbalance value compare to other schedulers. Although the MDTS agent is initially ignorant of the environment information, it has learned this knowledge entirely by those rewards that environment returns after each scheduling, and even performs better than PSO, the heuristic algorithm with an accurate model of the environment.

We have made various adjustments to the task scheduling environment to confirm that MDTS can maintain good performance in a variety of situations. In Fig. 6, all the x-axes represent the $TaskDemand$, which is changed by modifying the average job arrival interval. And in Fig. 7, we change the ratio of long tasks to all tasks and record the scheduling performance of each scheduler under various ratios. From Figs. 6-7, we can see that MDTS has unarguably delivered superior performance in varies situations. In other words, the MDTS agent can always learn suitable scheduling strategies from a variety of complex environments and continuously improve them through experience.

Moreover, it is noted that the performance of MDTS fluctuates in Figs. 4(c), Figs. 6(c), and Figs. 7(c). At the beginning of

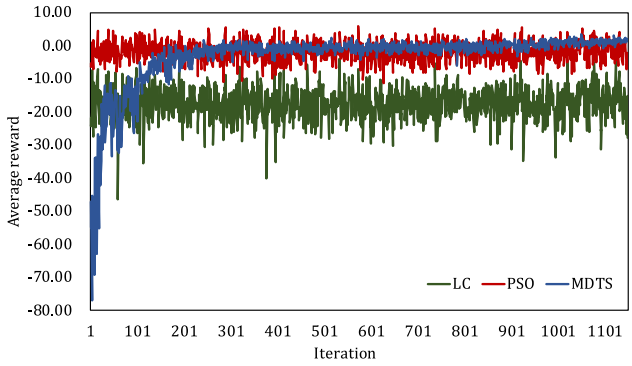

Fig. 5. Average job execution time over 1000 iterations.

this section, the parameters are set to: $\lambda := 6, \mu := 1, \nu := 3$, in order to consider multiple scheduling indicators simultaneously and focus more on shortening the average job completion time. Therefore, as the computing demand for each job (number of tasks in one job, task demand, proportion of long tasks) increases, MDTS will consider assigning more tasks to nodes with powerful computing capability but higher cost, sacrificing some reward of the cost part to ensure the reward of average execution time part, which makes the performance of MDTS fluctuant. While for LC, the total cost shows a clear trend of growth with the computing demand increasing because it only considers the number of connections when scheduling and cannot fully utilize the resources of those computing nodes with lower cost.

A comparative experiment is conducted to explain whether MDTS performs better than other DRL algorithms, such as the original A3C and DQN [27]. Since the original DRL algorithm could not handle the explosive action space, we fix the number of tasks to three and set all the task allocation schemes to the action space whose size is 1000. The scheduling scheme is selected according to the output probability. Nevertheless, it is still difficult to converge for DQN, so we have to split

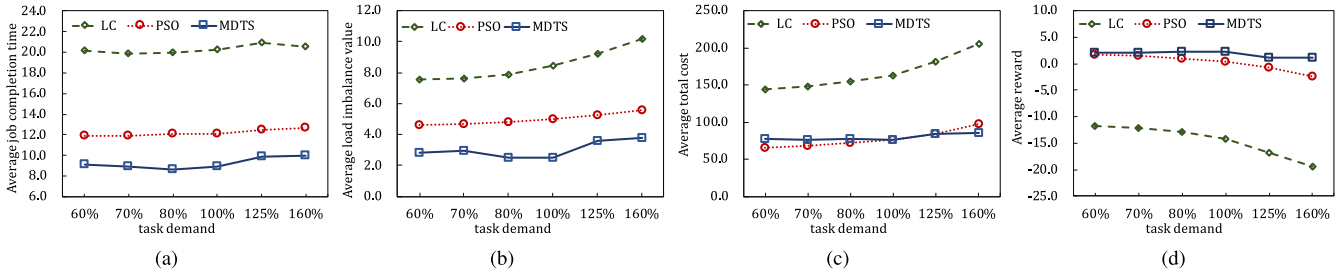

Fig. 6. Average job completion time (a), load imbalance value (b), total cost (c), and overall reward (d) under various $TaskDemand$.

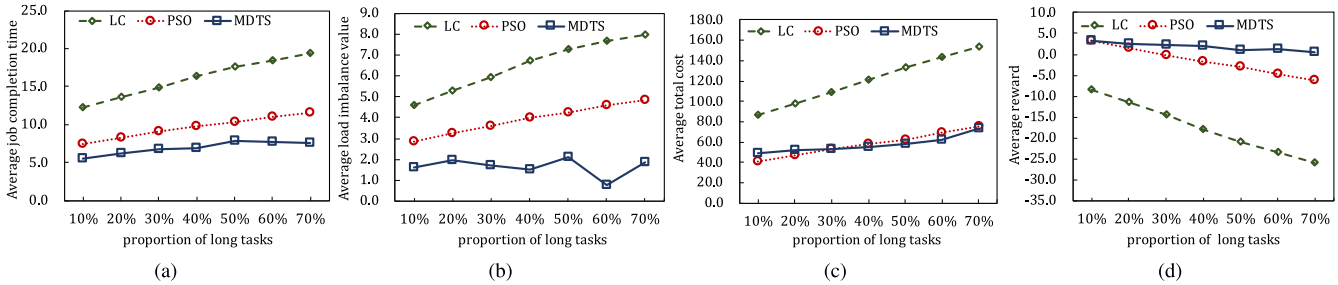

Fig. 7. Average job completion time (a), load imbalance value (b), total cost (c), and overall reward (d) under various ratio of long tasks to all tasks.

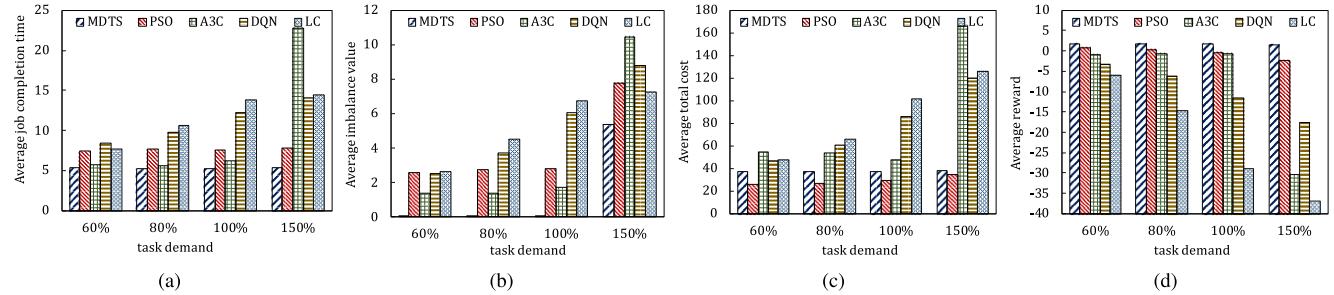

Fig. 8. Comparison of average job completion time (a), load imbalance value (b), total cost (c), and overall reward (d) under various task demands when the task number is fixed to 3.

the allocation scheme of one job into multiple decisions, and output only one allocation node of a child task once. Fig. 8 shows the comparison of the value of task scheduling indicators of MDTS, A3C, DQN, PSO and LC. The baselines of all scheduling objectives are accordingly changed: $T_{base} = 7, Load_{base} = 2.5, TotalCost_{base} = 30$. It demonstrates that MDTS does improve the performance of scheduling compared with A3C and DQN. According to the overall reward index, MDTS consistently maintains the highest scheduling performance. In detail, A3C gets short job execution time to some extent, but its overall reward is still negative due to high cost. Especially when task demand reaches 150%, A3C is difficult to complete learning tasks and no longer converges. These data prove that MDTS has learned the correlation among parallel tasks and has made good use of this knowledge when developing scheduling strategies. In order to make the size of action space acceptable for A3C algorithm, the number of parallel tasks is set too small. This design manifests the advantages of PSO, but MDTS still performs better on most of the scheduling metrics compared to other algorithms.

VI. Conclusion

In this paper, we studied the problem of scheduling jobs with scalable parallel tasks in VANETs, where there is a demand to determine the task placement plan with the goal of minimizing job completion time, load imbalance value, and total cost by making good use of the computing resources on vehicles and other edge computing nodes. Considering the complexity and dynamics of the vehicular network environment, a novel DRL-based scheduling approach was proposed in this paper, which can learn to assign tasks to appropriate distributed nodes solely from its experience rather than an accurate mathematical model. Moreover, encouraged by multi-task learning, we improved the neural network structure of DRL with multiple output branches, which realizes a fine-matched approach of

node assignments for parallel tasks. The MDTS completely considers the competition of resources among parallel tasks and achieves excellent scheduling effects. The performance of MDTS was evaluated by comparing them with other popular schedulers through simulation experiments. The results show that the MDTS improves overall reward, reduces the job completion time and optimizes the load balancing problem of a VANET and total costs of job execution simultaneously under various task demand. Experimental data show that MDTS can learn to reserve certain computing resources for future complicated jobs and choose nodes with stronger computing ability. Furthermore, it is also confirmed that the MDTS is superior to the raw DRL algorithms.

REFERENCES

[1] X. Hong, J. Jiao, A. Peng, J. Shi, and C. Wang, "Cost optimization for on-demand content streaming in IoV networks with two service tiers," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 38–49, Feb. 2019.

[2] Y. Ni, J. He, L. Cai, J. Pan, and Y. Bo, "Joint roadside unit deployment and service task assignment for Internet of Vehicles (IoV)," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3271–3283, Apr. 2019.

[3] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on IaaS cloud systems," *Int. J. Comput. Sci. Mobile Comput.*, vol. 2, no. 7, pp. 666–677, 2013.

[4] S. Devipriya and C. Ramesh, "Improved max-min Heuristic model for task scheduling in cloud," in *Proc. Int. Conf. Green Comput. Commun. Conservation Energy*, Chennai, India, Dec. 2013, pp. 883–888.

[5] D. Choi, K. Chung, and J. G. Shon, "An improvement on the weighted least-connection scheduling algorithm for load balancing in web cluster systems," in *Proc. Int. Conf. Grid Distrib. Comput. Control Automat.*, Jeju Island, Korea, Dec. 2010, pp. 127–134.

[6] X. Zuo, G. Zhang, and T. Wei, "Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 564–573, Apr. 2014.

[7] F. Pop, V. Cristea, N. Bessis, and S. Sotiriadis, "Reputation guided genetic scheduling algorithm for independent tasks in inter-clouds environments," in *Proc. Int. Conf. Adv. Inf. Netw. Appl. Workshops*, Barcelona, Spain, Mar. 2013, pp. 772–776.

[8] M. A. Tawfeek, A. El-Sisi, A. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," *Int. Arab J. Inf. Technol.*, vol. 12, no. 2, pp. 129–137, 2015.

[9] I. Hosu and T. Rebedea, "Playing atari games with deep reinforcement learning and human checkpoint replay," in *Proc. Eur. Conf. Artif. Intell. (ECAI) Workshop EGPAI*, Aug. 2016.

[10] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan, "DSCNN: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *Proc. IEEE Int. Conf. Comput. Des.*, Scottsdale, AZ, USA, Oct. 2016, pp. 678–681.

[11] R. Caruana, *Multitask Learning*. Boston, MA: Springer US, 1998, pp. 95–133.

[12] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, pp. 1633–1685, 2009.

[13] D. Zeng, G. Lin, G. Song, Z. Cheng, and Y. Shui, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3702–3712, Dec. 2016.

[14] M. Hu, J. Luo, W. Yang, and B. Veeravalli, "Adaptive scheduling of task graphs with dynamic resilience," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 17–23, Jan. 2017.

[15] G. Calinescu, C. Fu, M. Li, W. Kai, and C. J. Xue, "Energy optimal task scheduling with normally-off local memory and sleep-aware shared memory with access conflict," *IEEE Trans. Comput.*, vol. 67, no. 8, pp. 1121–1135, Aug. 2018.

[16] L. He, D. Zou, Z. Zhang, C. Chao, J. Hai, and S. A. Jarvis, "Developing resource consolidation frameworks for moldable virtual machines in clouds," *Future Gener. Comput. Syst.*, vol. 32, no. 1, pp. 69–81, 2014.

[17] X. Zhu, C. Chen, L. T. Yang, and Y. Xiang, "Angel: Agent-based scheduling for real-time tasks in virtualized clouds," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3389–3403, Dec. 2015.

[18] X. Mao, C. Li, W. Yan, and S. Du, "Optimal scheduling algorithm of MapReduce tasks based on QoS in the hybrid cloud," in *Proc. Int. Conf. Parallel Distrib. Comput. Appl. Technol.*, Guangzhou, China, Dec. 2016, pp. 119–124.

[19] A. Wilczynski and J. Kolodziej, "Modelling and simulation of security-aware task scheduling in cloud computing based on Blockchain technology," *Simul. Model. Pract. Theory*, vol. 99, p. 102038, 2020.

[20] H. F. Sheikh, I. Ahmad, and D. Fan, "An evolutionary technique for performance-energy-temperature optimized scheduling of parallel tasks on multi-core processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 3, pp. 668–681, Mar. 2016.

[21] R. Bleuse, S. Hunold, S. Kedad-Sidhoum, F. Monna, G. Mounie, and D. Trystram, "Scheduling independent moldable tasks on multi-cores with gpus," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2689–2702, Sep. 2017.

[22] R. M. Pathan, P. Voudouris, and P. Stenstrom, "Scheduling parallel real-time recurrent tasks on multicore platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 4, pp. 915–928, Apr. 2018.

[23] N. Liu, M. Liu, W. Lou, G. Chen, and J. Cao, "PVA in VANETs: Stopped cars are not silent," in *Proc. IEEE Int. Conf. Comput. Commun.*, Shanghai, China, Apr. 2011, pp. 431–435.

[24] C. Wang, L. Yong, D. Jin, and C. Sheng, "On the serviceability of mobile vehicular cloudlets in a large-scale urban environment," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 10, pp. 2960–2970, Oct. 2016.

[25] C. Xu, J. Lei, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2015.

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

[27] Y. Wang, K. Wang, H. Huang, T. Miyazaki, and S. Guo, "Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 976–986, Feb. 2019.

[28] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[29] S. Shen, Y. Han, X. Wang, and Y. Wang, "Computation offloading with multiple agents in edge-computing-supported IoT," *ACM Trans. Sen. Netw.*, vol. 16, no. 1, pp. 1–27, Dec. 2019.

[30] Y. Liu, S. Wang, J. Huang, and F. Yang, "A computation offloading algorithm based on game theory for vehicular edge networks," in *Proc. IEEE Int. Conf. Commun.*, Kansas City, MO, USA, May 2018, pp. 1–6.

[31] X. Huang, P. Li, and R. Yu, "Social welfare maximization in container-based task scheduling for parked vehicle edge computing," *IEEE Commun. Lett.*, vol. 23, no. 8, pp. 1347–1351, Aug. 2019.

[32] A. Kontarinis, V. Kantere, and N. Koziris, "Cloud resource allocation from the user perspective: A bare-bones reinforcement learning approach," in *Proc. Int. Conf. Web Inf. Syst. Eng.*, 2016, pp. 457–469.

[33] X. Lin, Y. Wang, and M. Pedram, "A reinforcement learning-based power management framework for green computing data centers," in *Proc. IEEE Int. Conf. Cloud Eng.*, Berlin, Germany, Apr. 2016, pp. 135–138.

[34] B. Mao *et al.*, "Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1946–1960, Nov. 2017.

[35] B. Mao *et al.*, "A novel non-supervised deep-learning-based network traffic control method for software defined wireless networks," *IEEE Wireless Commun.*, vol. 25, no. 4, pp. 74–81, Aug. 2018.

[36] B. Mao, F. Tang, Z. M. Fadlullah, and N. Kato, "An intelligent route computation approach based on real-time deep learning strategy for software defined communication systems," *IEEE Trans. Emerg. Topics Comput.*, pp. 1–1, 2019, doi: 10.1109/TETC.2019.2899407.

[37] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM Workshop Hot Topics Netw.*, Atlanta, GA, USA, Nov. 2016, pp. 50–56.

[38] M. Cheng, J. Li, and S. Nazarian, "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *Proc. Asia South Pacific Des. Automat. Conf.*, Jeju, Korea (South), Jan. 2018, pp. 129–134.

[39] Y. Wang*et al.*, "Multi-objective workflow scheduling with deep-Q-network-based multi-agent reinforcement learning," *IEEE Access*, vol. 7, pp. 39 974–39 982, 2019.

[40] H. Lu, C. Gu, F. Luo, W. Ding, and X. Liu, "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 102, pp. 847–861, 2020.
[41] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.
[42] Z. Ning *et al.*, "Deep reinforcement learning for intelligent Internet of Vehicles: An energy-efficient computational offloading scheme," *IEEE Trans. Cogn. Commun. Netw.*, vol. 5, no. 4, pp. 1060–1072, Dec. 2019.
[43] M. Zhou, Y. Yu, and X. Qu, "Development of an efficient driving strategy for connected and automated vehicles at signalized intersections: A reinforcement learning approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 1, pp. 433–443, Jan. 2020.
[44] Y. Sun *et al.*, "Adaptive learning-based task offloading for vehicular edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3061–3074, Apr. 2019.
[45] X. Pham, T. Nguyen, V. Nguyen, and E. Huh, "Joint node selection and resource allocation for task offloading in scalable vehicle-assisted multi-access edge computing," *Symmetry*, vol. 11, no. 1, pp. 1–17, 2019.
[46] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2322–2358, Oct./Dec. 2017.
[47] V. Mnih *et al.*, "A synchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, New York City, NY, USA, Jun. 2016, pp. 1928–1937.
[48] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "PAC model-free reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, vol. 148. Pittsburgh, Pennsylvania, USA: ACM, Jun. 2006, pp. 881–888.
[49] N. Liu *et al.*, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, Atlanta, GA, USA, Jun. 2017, pp. 372–382.
[50] E. Brunskill and L. Li, "Sample complexity of multi-task reinforcement learning," in *Proc. Conf. Uncertainty Artif. Intell.* Bellevue, WA, USA: AUAI Press, Aug. 2013, arXiv:1309.6821.

Qi Qi received the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2010. She is currently an Associate Professor of the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She has authored or coauthored more than 30 papers in international journal, and obtained two National Natural Science Foundations of China. Her research interests include edge computing, mobile cloud computing, Internet of Things, ubiquitous services, deep learning, and deep reinforcement learning.

Lingxin Zhang received the B.S. degree in 2018 from the Beijing University of Posts and Telecommunications, Beijing, China, where she is currently working toward the master's degree with the State Key Laboratory of Networking and Switching Technology. Her research interests include deep reinforcement learning, MDP, deep learning, parallel computing, and resource management.

Jingyu Wang received the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2008. He is currently an Associate Professor of the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He has authored or coauthored more than 50 papers in international journal, including the *IEEE Communication Magazine*, IEEE SYSTEM, and so on. His research interests include span broad aspects of SDN, big data processing and transmission, overlay networks, multi-media services, and traffic engineering.

Haifeng Sun received the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2017. He is currently a Lecturer of the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interests include span broad aspects of AI, NLP, big data analysis, object detection, deep learning, deep reinforcement learning, SDN, processing.

Zirui Zhuang is currently working toward the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His research interests include network routing and management for next-generation network infrastructures, using machine learning and artificial intelligence techniques, including deep learning, reinforcement learning, graph representation, multi-agent system, and Lyapunov-based optimization.

Jianxin Liao received the Ph.D. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 1996. He is currently the Dean of Network Intelligence Research Center and a Full Professor of the State Key laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. He has authored or coauthored hundreds of research papers and several books. He has won a number of prizes in China for his research achievements, which include the Premier's Award of Distinguished Young Scientists from the National Natural Science Foundation of China in 2005, and the Specially-invited Professor of the "Yangtse River Scholar Award Program" by the Ministry of Education in 2009. His main research interests include cloud computing, mobile intelligent network, service network intelligent, networking architectures and protocols, and multimedia communication.

F. Richard Yu (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the University of British Columbia (UBC), Vancouver, BC, Canada, in 2003. From 2002 to 2006, he was with Ericsson, in Lund, Sweden and a startup in California, USA. In 2007, he joined Carleton University, Ottawa, ON, Canada, where he is currently a Professor. He received the IEEE TCGCC Best Journal Paper Award in 2019, Distinguished Service Awards in 2019 and 2016, Outstanding Leadership Award in 2013, Carleton Research Achievement Award in 2012, the Ontario Early Researcher Award (formerly Premiers Research Excellence Award) in 2011, the Excellent Contribution Award at IEEE/IFIP TrustCom 2010, the Leadership Opportunity Fund Award from Canada Foundation of Innovation in 2009 and the Best Paper Awards at IEEE ICNC 2018, VTC 2017 Spring, ICC 2014, Globecom 2012, IEEE/IFIP TrustCom 2009 and International Conference on Networking 2005. His research interests include connected/autonomous vehicles, security, artificial intelligence, distributed ledger technology, and wireless cyber-physical systems. He serves on the editorial boards of several journals, including Co-Editor-in-Chief for *Ad Hoc & Sensor Wireless Networks*, Lead Series Editor for the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE COMMUNICATIONS SURVEYS & TUTORIALS, and IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING. He has served as the Technical Program Committee (TPC) Co-Chair of numerous conferences. He is a registered Professional Engineer in the province of Ontario, Canada, IET Fellow, and Engineering Institute of Canada (EIC) Fellow. He is an IEEE Distinguished Lecturer of both Vehicular Technology Society (VTS) and Communication Society. He is an elected member of the Board of Governors of the IEEE VTS.