sensors

MDPI

Article

Software-Defined Optimal Computation Task Scheduling in Vehicular Edge Networking [†]

Zhiyuan Li [1,2,3,*] and Ershuai Peng [1]

[1] College of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China; 2221808035@stmail.ujs.edu.cn
[2] Jiangsu Province Big Data Ubiquitous Perception and Intelligent Agricultural Application Engineering Research Center, Zhenjiang 212013, China
[3] Jiangsu Key Laboratory of Security Technology for Industrial Cyberspace, Zhenjiang 212013, China
[*] Correspondence: lizhiyuan@ujs.edu.cn; Tel.: +86-187-9600-6616

[†] This paper is an extended version of our paper published in Peng, E.; Li, Z. Optimal control-based computing task scheduling in software-defined vehicular edge networks. In Proceedings of the IEEE International Conference on Internet of Things and Intelligent Applications (ITIA), Zhenjiang, China, 27–29 November 2020.

Abstract: With the development of smart vehicles and various vehicular applications, Vehicular Edge Computing (VEC) paradigm has attracted from academic and industry. Compared with the cloud computing platform, VEC has several new features, such as the higher network bandwidth and the lower transmission delay. Recently, vehicular computation-intensive task offloading has become a new research field for the vehicular edge computing networks. However, dynamic network topology and the bursty computation tasks offloading, which causes to the computation load unbalancing for the VEC networking. To solve this issue, this paper proposed an optimal control-based computing task scheduling algorithm. Then, we introduce software defined networking/OpenFlow framework to build a software-defined vehicular edge networking structure. The proposed algorithm can obtain global optimum results and achieve the load-balancing by the virtue of the global load status information. Besides, the proposed algorithm has strong adaptiveness in dynamic network environments by automatic parameter tuning. Experimental results show that the proposed algorithm can effectively improve the utilization of computation resources and meet the requirements of computation and transmission delay for various vehicular tasks.

Keywords: software-defined vehicular edge networking; resource allocation; computation task scheduling; optimal control

check for updates

Citation: Li, Z.; Peng, E. Software-Defined Optimal Computation Task Scheduling in Vehicular Edge Networking . *Sensors* **2021**, *21*, 955. https://doi.org/10.3390/s21030955

Academic Editor: Lei Shu

Received: 5 January 2021
Accepted: 28 January 2021
Published: 1 February 2021

1. Introduction

With the development of the intelligent connected vehicles, more and more vehicular terminal devices have begun to participate in data task collection and processing [1–4], such as autonomous driving, virtual reality, and computation-intensive tasks. [5] presented a traffic management service, called ABATIS, to optimize the routes for vehicles. In cloud computing paradigm, the collected data and processing tasks have been uploaded to a centralized data center far away from Internet of vehicles [6]. After the task execution is completed, the processing results will be transmitted back to the task initiator. Hence, enormous sensed data and the encapsulated computation-intensive tasks transmission have put severe strain on the mobile network, especially in terms of bandwidth consumption. With the ever-increasing amount of network traffic, the network congestion frequently happens and degrades the quality of service (QoS) for delay-sensitive tasks [7–9].

Recently, a Vehicular Edge Computing (VEC) paradigm was presented, as shown in Figure 1. A vehicle can offload some complex delay-sensitive tasks to edge cloud due to the limited computing, storage, communication capacity [8,10,11], as shown in

Figure 2. There are existing works focusing on the task offloading and resource allocation. For example, Chen [12] exploits Lyapunov optimization theory to maximize the long-term performance of networking system, such as lower task execution delay and task migration delay. Although Lyapunov optimization algorithm can be designed and implemented, it is not suitable to complex and highly dynamic network environments. Literature [13] proposed the Cloud Computing Management Unit (CCMU) to optimize the allocation of computing resources. In [13], CCMU uses the Markov decision process to give the optimization decisions. Although the MDP has been developed a general algorithm (Bellman Equation), the convergence speed of MDP is still slow for the state space of continuous variables. Overall, it is difficult to achieve the load balancing effect for the existing task offloading and resource allocation algorithms. How to reasonably allocate computing and communication resources is still a challenge for the VEC networking. In our previous research, our work is the exploiting the modern control theory to model the computation task optimal scheduling in vehicular edge networks [14]. This paper is the extension of the conference paper [14]. In this paper, we proposed an Optimal Control-based computing Task Scheduling algorithm (OCTS). The proposed OCTS method has significantly improved the previous work in the following aspects: (1) Introducing VEC architecture and assumptions; (2) using the software-defined network (SDN)/OpenFlow to collect the necessary network parameters for task scheduling; (3) optimizing task migration delay and avoiding network congestion by adjusting the weight value ξ. When the network delay is higher than the threshold value, it can be reduced by increasing weight value ξ; (4) enhancing the experimental verification; and (5) comparisons with the existing approaches.

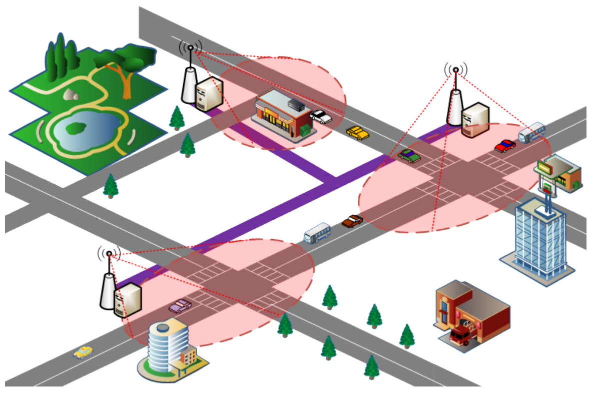

Figure 1. Vehicular edge computing paradigm.

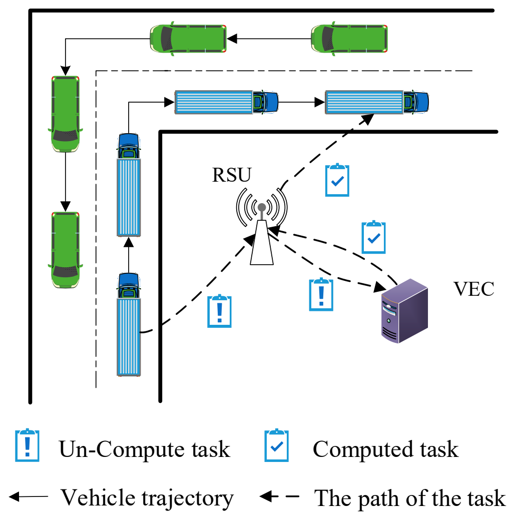

Figure 2. Illustration of Vehicular Edge Computing (VEC) network application scenario.

The remainder of this paper is organized as follows. Section 2 gives the related work. Section 3 describes the network scenario and system model. Section 4 presents the vehicular task queuing model. Section 5 proposes the optimal control-based computing task scheduling algorithm. Section 6 presents our empirical studies. Section 7 concludes this paper.

2. Related Works

With the development of Internet of vehicles, more and more vehicular applications have been developed to meet the QoS requirements of mobile users. Due to the limited computing, storage, and communication capacities of mobile terminals, the vehicle itself cannot be able to meet the QoS requirements of various applications. Although cloud computing has large computing power, the transmission delay between vehicles and cloud computing platform is significant [15]. Hence, it is not sufficient for the latency-sensitive applications. The edge computing servers are closer to the mobile terminals, which can effectively reduce task execution and migration delay [8].

Nowadays, some researchers have focused on the mobile task offloading and resource allocation. A previous study [16] proposed a joint optimal VEC server selection and offloading (JSCO) Algorithm to address resource allocation for a multiuser multi-server VEC system. Fan [17] proposed a collaborative optimization migration and caching model to improve the performance of edge task execution. Then, the optimization problem with two independent sub-problems was solved. Next, the resource management algorithm was designed to jointly schedule task and its migration. The experimental results show that the proposed model can reduce the latency of task execution. Another study [18] studied the average delay minimization problem for component-based linear applications in vehicular ad-hoc networks and proposed a delay-optimization based ant colony optimization algorithm (DoACO). This optimization problem with the stringent constraint conditions has been proved a NP hard problem. Liu [19] presented the optimization problem with the energy consumption, execution delay, and price cost constraints. Specifically, energy consumption, execution delay, and computing capacity were explicitly and jointly considered. On the basis of theoretical analysis, a multi-objective optimization problem with joint conditions was presented. To minimize the power consumption, execution delay, and price cost, the multi-objective optimization problem was solved by the scaling scheme and the interior point method. Another study [20] proposed a resource-sharing scheme for data center collaboration, which stipulated that each data center uses buffer to store service requests for local execution. When the buffer is full, the request is migrated to an adjacent data center and accepted if the current queue length is below the pre-defined threshold in the data center. In this way, the blocking state and task execution latency of the data center can be effectively reduced. Furthermore, previous research [21] studied three load sharing schemes, namely no-sharing, random sharing, and minimum load sharing. After the comparisons, they find that the minimum load sharing scheme is most suitable for making full use of the cooperation among servers to realize the load balancing. Another study [22] proposed another edge computing task scheduling model, which transformed the waiting time minimization problem into an overall planning problem, and then carried out optimal scheduling through dynamic programming. A previous study [23] proposed an improved chaotic bat swarm algorithm. Based on the bat algorithm, chaos factors and second-order oscillation were introduced to accelerate the update of dynamic parameters and thus improve the convergence of the algorithm.

However, the proposed methods [12,20–28] result in unbalancing resource allocation for vehicular tasks due to dynamic network environment driven by humans. In order to achieve the load-balancing, literatures [29,30] used the idea of software defined network to obtain the more network status parameters. SDN is an innovative network design, implementation, and management method, which separates network control from forwarding process to achieve better user experience [31]. Next, the collected network status parameters are the input vector of the proposed optimization model. Experiments show that the proposed algorithms can significantly reduce task execution time. Although the proposed algorithms can achieve the better results, the number of iterations being solved is still large due to the high dimension input vector. Hence, the presented solving algorithms are not fundamentally suitable for delay-sensitive vehicular task execution.

3. VEC Architecture and Assumptions

In this section, we first introduce the networking layer architecture. Then, the software-defined vehicular edge networking (SD-VEC) architecture and computation models are elaborated.

3.1. SD-VEC Networking Layer Architecture

Figure 3 shows the SD-VEC networking layer architecture. In the user layer, when the vehicles in the area covered by a Roadside Unit (RSU), they can send vehicular tasks to the RSU with wireless connections. In the VEC layer. Each VEC server is connected with an RSU. The RSU will forward these tasks to the VEC servers and then the VEC server will execute these tasks and send them back to the requesting vehicles. The VEC servers in different area are connected by the wired cables in the same local area network. The VEC servers can send or receive tasks to or from other servers by these wired cables. In the control layer, the SDN controller can connect with the VEC servers through wired cables. The SDN controller can not only obtain the status information of servers and network, such as the CPU utilization, memory usage, but also control the task migration among the servers. The descriptions of symbols are shown in Table 1.

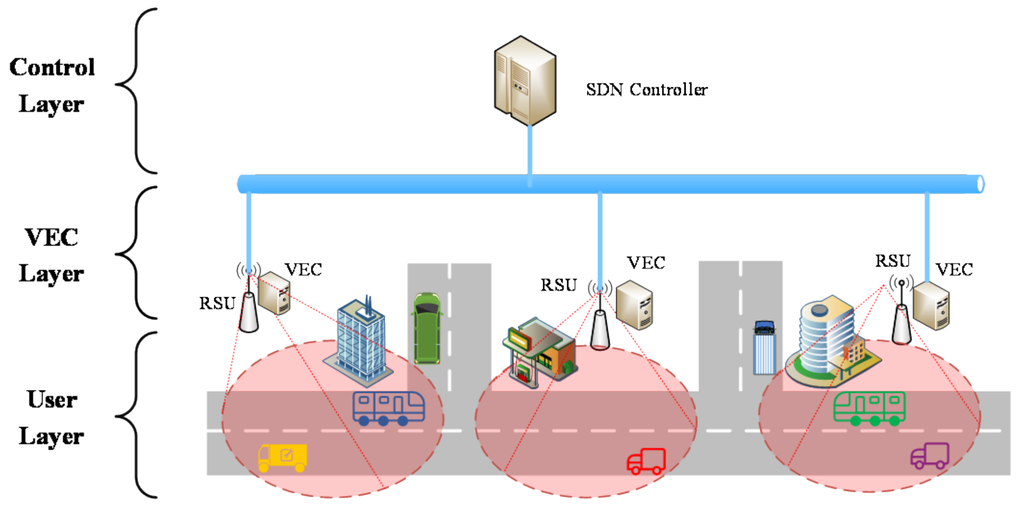

Figure 3. Software-defined vehicular edge networking (SD-VEC) networking layer architecture.

3.2. Network Model Assumptions

We assume that the second layer includes N VEC servers, which are deployed near the RSU. Each VEC server is equipped with an RSU thus the VEC servers and RSUs share the same index. When the vehicles move into the coverage of the RSUs, they can offload their computation tasks to corresponding RSUs via wireless link. The CPU frequency of a VEC server i is denoted as f_i and the corresponding clock period is $h_i = 1/f_i$. All of the CPU frequencies for VEC servers can be denoted as $f = \{f_1, \cdots\cdots, f_i\}$ and the corresponding CPU clock periods can be denoted as $h = \{h_1, \cdots\cdots, h_i\}, i \in N$.

Secondly, we assume that the received tasks by server i follow a Poisson distribution. We denote $\lambda_i(t)$ as the number of the tasks arriving at the VEC server i at time t, and the task execution time on server i at time t is denoted as $m_i^t. = \rho_i^\lambda(t)$ is denoted as the processing time to run tasks on a server i. All tasks arriving at the VEC servers are denoted as $\lambda(t) \in \{\lambda_1(t), \cdots\cdots, \lambda_i(t)\}, i \in N$ and their corresponding processing time can be denoted as $\rho^{\lambda}(t) = \{\rho_1^{\lambda}(t), \cdots\cdots, \rho_i^{\lambda}(t)\}, i \in N$. To simplify the computation, we assume that the task execution time is $m_i^t \times h_i$ for a VEC serve i. Hence, the server i can take $h_i \times m_i^t \times \lambda_i(t)$ to process the tasks at time t.

Table 1. Description of the symbols.

Symbols	Description
f_i	The CPU frequency of VEC server i
h_i	The CPU clock period of VEC server i
$\lambda_i(t)$	The number of the tasks arriving at VEC server i at time t
m_i^t	The task execution time on server i at time t
ρ_i^{λ}	The processing time to run tasks on server i
$u_{ij}(t)$	The number of the tasks sent form server i to server j
τ	The task transmission time with no network congestion
ND_i	The task transmission time on server i
CD_i	The task execution time on server i
x_i	State variable for VEC server i
J	The convergence cycle of the system at time t
$\boldsymbol{x}^*(t)$	The optimal routing trajectory at time t
$\boldsymbol{u}^*(t)$	The optimal control vector at time t
$C_{\max}$	Threshold value of CPU utilization
φ	The optimum range of CPU utilization for a VEC server
G	The vector form of φ
D	The execution time and transmission time incurred by offloaded task scheduling
ξ	A coefficient which is used to adjust the weight of the offloaded task transmission time

Thirdly, during the work of this system, the network delay cannot be easily calculated. In order to adjust and optimize the task schedule via the network state, we use the SDN to obtain the network state. The network delay obtained by SDN is defined as D^a, and the maximum network delay defined by a user is defined as $D^a_{\max}$.

4. Vehicular Task queuing Model

Due to the limited computation capacity of a VEC server and the burst of task arrivals, some VEC servers are busy, while others are free. The load imbalance will lead to a low computation resources utilization and extra computation time. Hence, we have to schedule tasks among VEC servers to alleviate the load imbalance.

We assume that we send tasks of server i to server j and the number of the tasks can be denote as $u_{ij}(t)$. Specifically, if server i needs to process a large number of tasks at time t while server j has to process a small number of tasks, the $u_{ij}(t)$ could be positive number. Thus, the number of tasks which server i send to others can be denoted as $u_{i\cdot}(t) = \sum_{j=1}^{N_{\max}} u_{ij}(t), j \in N$. According to the presented computation model, at time t the VEC server i will take $h_i \times m_i^t \times (\lambda_i(t) - u_{i\cdot}(t))$ to process these tasks. Vehicular task queuing model is shown as Figure 4. Next, we describe the task queuing model and task computation model.

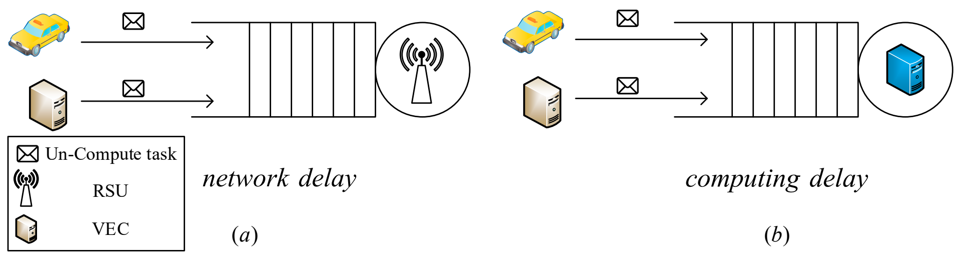

Figure 4. Vehicular task queuing model.

4.1. Task Queuing Model

Task scheduling not only can effectively decrease the task execution time of high load servers, but also can increase computation resource utilization. However, the task scheduling will increase the task transmission time of the VEC network. To calculate the task transmission time, we assume that the tasks arrive at server i from other servers follow a Poisson distribution. We also assume that the task transmission time is τ when there is no network congestion. As shown in Figure 4a, we use M/M/1 queuing model to build the vehicular task queuing model. The task transmission time can be denoted as Equation (1), where $u_i(t) = \sum_{j=1}^{N} u_{ji}(t)$.

$$ND_i(u_i(t)) = \frac{\tau}{1 - \tau u_i(t)} \tag{1}$$

4.2. Task Computation Model

As shown in Figure 4b, we use M/M/1 to build the vehicular task queuing model. The task execution time is denoted as Equation (2).

$$CD_i(u_i(t)) = \frac{1}{f_i/m_i^t - u_i(t)}, i \in N \tag{2}$$

where CD_i is denoted as task execution time on a server i.

5. Problem Formulation and Optimal Task Scheduling Solving

Our goal is to reduce the load on each server as much as possible while satisfying the network delay and computing delay of vehicular tasks.

The problem formulation with constraints is shown as Formula (3).

$$\begin{gathered} \min \boldsymbol{J}(\boldsymbol{u}(t)) = \int_0^T (\xi ND(\boldsymbol{u}(t)) + CD(\boldsymbol{u}(t)))dt \\ \text{s.t. } \boldsymbol{S} = \{\boldsymbol{x}(T) | \boldsymbol{G}(\boldsymbol{x}(t)) = 0\} \\ ND_i(u_{\cdot t}(t)) \le ND_{\max}, \forall i \in N \\ CD_i(u_{\cdot t}(t)) \le CD_{\max}, \forall i \in N \end{gathered} \tag{3}$$

In the Formula (3),

$$\boldsymbol{J}(\boldsymbol{u}(t)) = \int_0^T (\xi ND(\boldsymbol{u}(t)) + CD(\boldsymbol{u}(t)))dt \tag{4}$$

is the performance index of the VEC system. $ND(\boldsymbol{u}(t))$ and $CD(\boldsymbol{u}(t))$ is the execution time and transmission time incurred by offloaded task scheduling. When $D^a > D_{\max^a}$, the increasing value of ξ is to decrease D^a. ξ is a coefficient to adjust the weights of the offloaded task transmission time. If we increase the value of ξ, the task transmission time will decrease while the task execution time will increase. The adjustment of the ξ value is used for dealing with the stringent time constraint vehicular tasks. Formula (3) can be resolved by calculus of variations method, but before that, we need to set up the corresponding state equations and state variables.

Firstly, the CPU utilization of a VEC server is denoted as a state variable. Next, the various tasks with different QoS requirements from both vehicles and other mobile terminals (laptop, smart phones, etc.) are denoted as the inputs of the system framework. The inputs of the network system have clearly influenced on running state variables of this system. The state variables of a VEC server i can be denoted as Equation (5).

$$x_i = \frac{h_i \times m_i^t \times (\lambda_i(t) + u_{i\cdot}(t))}{h_i \times m_i^t \times (\lambda_i(t) + u_{i\cdot}(t)) + \rho_i^\lambda(t)}, i \in N \tag{5}$$

Consequently, the running state vectors of the VEC system can be defined as Equation (6).

$$\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix}, i \in N \tag{6}$$

Secondly, the running state equation of the VEC system is used to describe the relationship between the input of this system and the system state. The running state equation of this system can be obtained, as shown in Equation (7).

$$\dot{\boldsymbol{x}} = \frac{d\boldsymbol{X}}{dt} \tag{7}$$

Furthermore, Formula (7) can be equivalently transformed into Formula (8).

$$\dot{\boldsymbol{x}} = \boldsymbol{A}(t)\boldsymbol{x}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t) \tag{8}$$

where the matrix $\boldsymbol{A}(t)$ represents the relationship among the state variables within the VEC system. The matrix $\boldsymbol{B}(t)$ represents the state control variable, which is used to control and track the variations of running states.

The CPU utilization over time can be represented by the vector $\boldsymbol{x}(t)$. The vector $\boldsymbol{x}(t)$ is referred as the trajectory of the VEC system. The scheduled tasks among the VEC servers can be denoted as $\boldsymbol{u}(t)$, namely the control vector. Through scheduling tasks among VEC servers, we can effectively control the CPU utilization of VEC servers from the initial running state to the final running state. In the entire running process, the offloaded task execution time and migration time can be controlled in an optimum range during the convergence cycle. Then, the convergence cycle of the VEC system can be denoted as a functional variable $\boldsymbol{J}$. If we get the minimum value of the vector $\boldsymbol{f}$ by using control vector $\boldsymbol{u}(t)$ to control vector $\boldsymbol{x}(t)$ from the original state $\boldsymbol{x}(0)$ to the destination state $\boldsymbol{x}(t_f)$, we could get an optimal routing trajectory for all offloaded tasks in the VEC system. Here, the optimal routing trajectory is denoted as $\boldsymbol{x}^*(t)$. t_f is the time consumption for the optimization process. The corresponding control vector is called optimal control vector, which is denoted as $\boldsymbol{u}^*(t)$.

Next, the status information of the servers, such as the CPU utilization and the network bandwidth, is collected by the SDN controller. The offloaded task among these servers is totally scheduled through the designed load-balancing app on a SDN controller. Optimized task scheduling among VEC servers can improve the load imbalance for the VEC virtual resources. Specifically, the VEC servers with higher load will migrate their tasks to that with lower load. Furthermore, the CPU utilization of all the optimized VEC servers will keep on optimum status level.

Finally, a threshold value $C_{\max}\%$ for CPU utilization is pre-defined. When the CPU utilization of a VEC server has just exceeded the pre-defined value, this time is set as the initial state indexed by 0. In addition, the CPU utilization of a VEC server is assumed as the optimization object. After time T, we can obtain the final optimization state. Hence, we can the optimal range for all of the VEC servers between the initial state and the final optimization state through tasks scheduling. Here, the optimum range of CPU utilization for a VEC server is denoted as $\varphi(\boldsymbol{x}(T), T) = 0$, and the vector form of $\varphi(\boldsymbol{x}(T), T) = 0$ is represented as $\boldsymbol{G}(\boldsymbol{x}(t)) = 0$ and the range of $\boldsymbol{x}(T)$ is set as $\boldsymbol{S} = \{\boldsymbol{x}(T) | \boldsymbol{G}(\boldsymbol{x}(t)) = 0\}$.

At last, the proposed Algorithm 1 is shown as below.

Algorithm 1. OCTS

```
**Input:** Task arrival numbers, CPU usage of all SBSs, Control parameter $\xi$, $\boldsymbol{J}(\boldsymbol{u}(t))$
**Output:** Optimal control $\boldsymbol{u}^*(t)$, optimal trajectory $\boldsymbol{x}^*(t)$
1. **For** $t = 0; t \leq T$ **do**
2. Calculate the expected CPU usage of SBSs $C_e\%$;
3. Calculate the fitted curve $\lambda(t)$ and $\rho^{\lambda}(t)$ based on task arrival numbers;
4. Establish the state vector $\dot{\boldsymbol{x}}$ based on step 2 and 3;
5. Solving the extreme value of $\boldsymbol{J}(\boldsymbol{u}(t))$;
6. Get $\boldsymbol{u}^*(t)$ and $\boldsymbol{x}^*(t)$;
7. Update the CPU usage of SBSs;
8. **End For**
9. **Return** $\boldsymbol{u}_1^*(t), \cdots, \boldsymbol{u}_t^*(t), \cdots, \boldsymbol{u}_T^*(t)$
```

6. Experiments

6.1. Experiment Setup

In this section, the simulation experiments are designed and implemented to evaluate the performance for the proposed optimal control-based computation task scheduling in software-defined vehicular edge networking in terms of CPU utilization and delay. The minimum configuration of a server is Core i3 CPU and 4G memory for performing the experiment in real time.

We have presented the software-defined vehicular edge networking (SD-VEC) environments, as shown in Figure 5. As shown in this figure, the vehicle nodes firstly communicate with its RSU node, and then the RSU node can obtain the flow tables through the RSU controller, the vehicle nodes can route by the flow rules at last. Here, we choose the floodlight v1.2 as the SDN controller. The combination of simulation of urban mobility (SUMO v1.8) with instant virtual network (Mininet v2.2) as the base software platform are installed on a computer equipped with i7 CPU and 8GB memory. SUMO can generate the mobility pattern of the vehicles that is used by the Mininet. We chose the city of Luxembourg in European cities as the simulation scenario. That is because Luxembourg SUMO Traffic Scenario is well-known and frequently used to evaluate the VANETs communication system. In this network experiment, there is an SDN controller, there are 5 VEC servers and 100 vehicles in a SD-VEC network. As shown in Figure 5, one VEC server can connect 20 vehicles at the beginning. To simplify the calculation, the computation capacity of every VEC server is set to 1GHz and the task transmission time in the LAN is set to 10 ms without network congestion. The event of vehicular task arrivals on a VEC server follows the Poisson distribution, and every task will take 10^6 CPU cycles. The captured network traffic in this paper is the time series with white Gaussian noise. The rest of simulation experimental parameters are shown in Table 2.

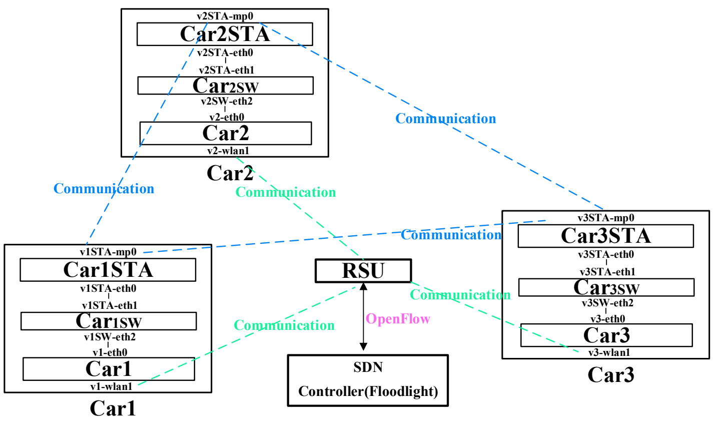

Figure 5. The communication architecture for SD-VEC.

Table 2. Description of the simulation experimental parameters.

Impact Factors on CPU Utilization	**CPU Utilization of Servers 1 to 5**				
	1	**2**	**3**	**4**	**5**
The first set of servers without any impact	20	20	30	30	80
The impact of Lyapunov on the first set of servers	33	33	35	35	48
The impact of OCTS on the first set of servers	35	35	37	37	45
The second set of serverswithout any impact	20	20	30	30	90
The impact of Lyapunov on second set of servers	33	33	35	35	53
The impact of OCTS on the second set of servers	35	35	39	39	49

6.2. Performance Evaluation

Figure 6 shows the network topology of the experiment. In the figure, the SDN Controller is connected with VEC by these cables, and the VECs is connected with the vehicles by wireless.

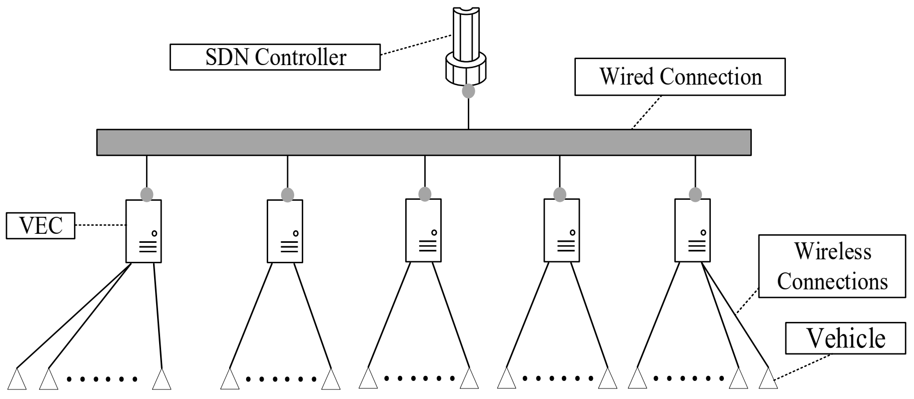

Figure 6. Network topology of the experiment.

As shown in Figure 7, the initial average CPU utilization of the edge servers 1 to 4 is 20% and CPU utilization of the edge server 5 is 90%. The initial average memory utilization of the edge servers 1 to 4 is 25% and memory utilization of the edge server 5 is 100%. The initial average disk utilization of the edge servers 1 to 4 is 23% and memory utilization of the edge server 5 is 93%. The total experiment duration is 1000 s. The experiments show that the usages of network load are uneven. Next, we simulated the Lyapunov optimization algorithm to improve the efficiency of the task scheduling on the edge servers. The average CPU utilization of servers 1 to 4 have increased from 20% to 30% and server 5 has decreased from 90% to 55%. The measurements of CPU utilization in this figure show the CPU utilization curves of the five edge servers are closer than that of without using any optimization. These results show the Lyapunov optimization algorithm can achieve the goal of computation load balancing. Then, we used the proposed OCTS optimization algorithm to schedule the offloaded task on the edge servers. As shown in Figure 7, the CPU utilization of servers 1 to 4 have increased from 20% to 35%, and the CPU usage of server 5 has decreased from 90% to 45%. The measurements of CPU utilization in Figure 7 show the CPU utilization curves of the five edge servers are closer than that of no optimization and using Lyapunov optimization. These experimental results prove that our proposed OCTS method is better than the Lyapunov optimization and no optimization methods in terms of computation load balancing. With the Lyapunov optimization, the average memory utilization of servers 1 to 4 have increased from 25% to 32% and server 5 has decreased from 100% to 65% while with the OCTS optimization, the average memory utilization of servers 1 to 4 have increased from 25% to 38% and server 5 has decreased from 100% to 53%. With the Lyapunov optimization, the average disk utilization of servers 1 to 4 have increased from 23% to 32% and server 5 has decreased from 93% to 63% while with the OCTS optimization, the average memory utilization of servers 1 to 4 have increased from 23% to 35% and server 5 has decreased from 93% to 50%.

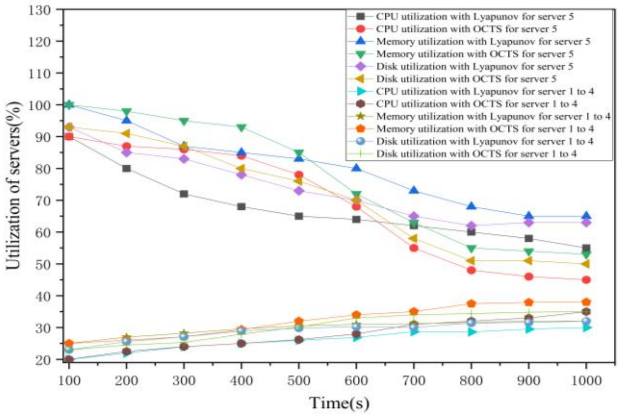

Figure 7. The edge server usage comparisons between Lyapunov optimization and our proposed method.

As shown in Figure 7, in the early stage of optimization, the effect of OCTS is not as good as that of Lyapunov. This is because if too many vehicular tasks are migrated at that time, there will be a high migration delay. When the load of the server drops, the optimization effect of OCTS is obviously better than that of Lyapunov, because when the computing delay of the server drops, the migration of a large number of vehicular tasks will not lead to too high migration delay.

Hence, we can get a conclusion that the optimization effect for the proposed OCTS algorithm is about 10% to 15% higher than the Lyapunov optimization method under the same status conditions. Furthermore, the optimization effect for the proposed OCTS algorithm is about 30% to 40% higher than no optimization method under the same status conditions.

Secondly, we compare the CPU utilization among the no optimization, the local optimization, and global optimization, as shown in Figure 8. Global optimization means that we can get load and delay information for each server through the SDN controller. In this figure, when the system runs without optimization, server 2 is about 20% CPU utilization and server 4 is about 30% CPU utilization. When the system is under local optimization, server 2 is about 20% CPU utilization and server 4 is about 45% CPU utilization. At this time, server 5 offloads a large number of tasks to server 4, and server 2 remains idle due to the lack of global performance status information. Tasks from server 5 is denied when server 4's CPU utilization is approaching the threshold.

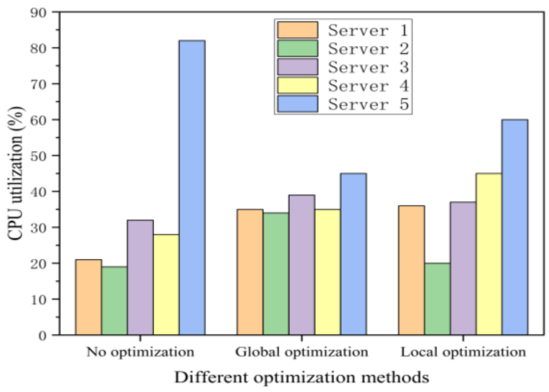

Figure 8. CPU utilization comparisons among the local optimization, no optimization, and global optimization methods.

Hence, when the system is under the global optimization, the CPU utilization of server 2 is about 33%, and that of server 4 is about 34%. The experimental results show that our proposed algorithm can achieve more excellent load balancing effect than another two methods.

Thirdly, we compare the no optimization with the OCTS optimization methods in the terms of the delay metric in unit of millisecond. As shown in Figure 9, the proposed OCTS method is compared with no optimization method in terms of both the task migration time and the task processing time. The experimental results show that both the task processing and task migration time for the servers 1 to 8 slightly increases when our proposed OCTS method enables. Besides, although the task migration time for servers 9 and 10 have also slightly increases, the task processing time of server 9 and 10 has obviously decreased. After the OCTS optimization finished, the task processing time and the task migration time among the edge servers have become approximately equal. The experimental results prove that our proposed OCTS method can obviously improve the computation and network load balancing. However, the task migration time needs to further optimize. In this evaluation, we set the parameter ξ equal to 1. Next, we try to adjust the value of parameter ξ to optimize the solving. The parameter ξ is used to adjust the proportion of network delay in the load balancing optimization process for reducing the task migration delay. That is because the improvement of load balancing needs too large a number of task migrations from one edge server to another edge server.

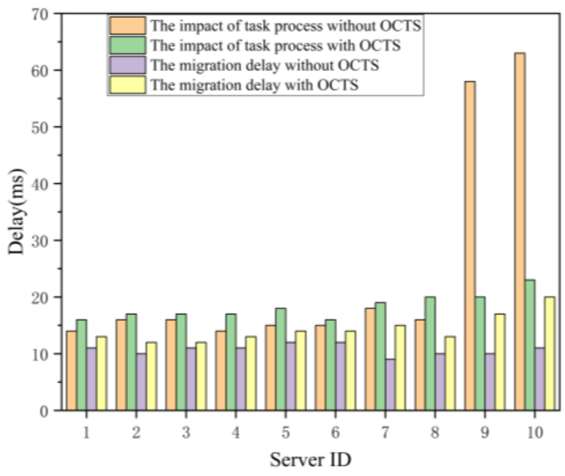

Figure 9. Delay variations on individual servers.

Fourthly, we study the parameter ξ optimization to reduce the task migration time. Here, we set a counter for an edge server to count the number of vehicular tasks arriving at the edge server. When the number of the offloaded task is up to 100, the counter will be reset to 0. Besides, the task execution time and task migration time will be calculated again. Then, we try to increase the value of parameter ξ to show the variations of task execution time and task migration time. As shown in Figure 10, compared with no optimization method, the experimental results show that both the task processing time and the task migration time of servers 1 to 8 decreases with the increasing of the value of ξ when our proposed OCTS method enables. Additionally, the task execution time of servers 9 and 10 are significantly reduced after the OCTS optimization. That is because a large number of tasks are scheduled from edge servers 9 and 10 to other edge servers. Meanwhile, the task migration time is doubled. That is because with the increasing value of parameter ξ, the OCTS optimization method appears over fit. As the experimental results show, we can come to the conclusion that there exists an optimal tradeoff between task execution time and task migration time when ξ is equal to 3.

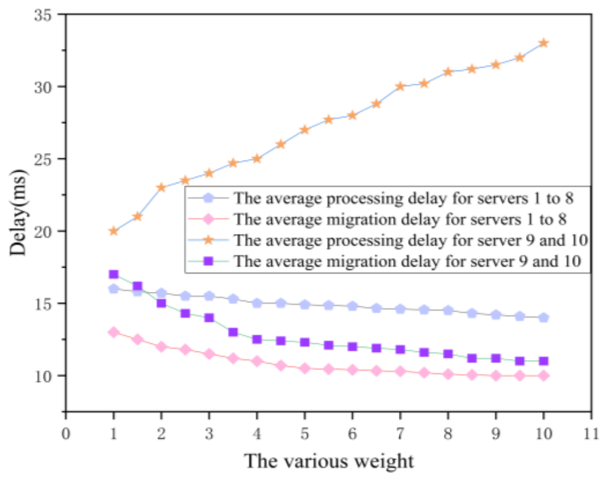

Figure 10. The impact on delay for servers 1 to 10.

7. Conclusions

In this paper, we propose an optimal control-based resource allocation algorithm, called OCTS, and exploit the software-defined networks framework to achieve the computation and network load balancing among the VEC servers in vehicular edge networking. Our contributions are that ① SD-VEC networking layer architecture is presented; ② the calculus of variations method in modern control theory is introduced to solve the optimal load balancing strategy for the offloaded edge computation tasks; ③ the parameter adjustment can optimize the task processing time and task migration time and meet the service requirements of vehicular users. Simulation experimental results prove that our proposed OCTS method can effectively reduce the load imbalance and improve the resource utilization of idle edge servers. Our future work is to study the user behavior and predict the load imbalance position in advance. According to the load utilization estimation, we can effectively schedule the vehicular tasks to the idle VEC servers.

Author Contributions: Conceptualization, Z.L.; Formal analysis, Z.L.; Funding acquisition, Z.L.; Investigation, Z.L.; Methodology, E.P.; Software, E.P.; Validation, E.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by National Key Research and Development Project under Grant 2017YFB1400703 and 2020YFB1005503, National Natural Science Foundation of China under Grant U1736216 and 61702233, by Natural Science Foundation of Jiangsu Province under Grant BK20201415, by China Postdoctoral Science Foundation of China under Grant 2015M570469.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Linthicum, D. Responsive data architecture for the internet of things. *Computer* **2016**, *49*, 72–75. [CrossRef]
2. Lin, J.; Yu, W.; Zhang, N.; Yang, X.; Zhang, H.; Zhao, W. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet Things J.* **2017**, *4*, 1125–1142. [CrossRef]
3. Li, Z.; Chen, R.; Liu, L.; Min, G. Dynamic resource discovery based on preference and movement pattern similarity for large-scale social internet of things. *IEEE Internet Things J.* **2016**, *3*, 581–589. [CrossRef]
4. Jiang, Y.; Wang, L.; Chen, H.-H. Covert communications in D2D underlaying cellular networks with antenna array assisted artificial noise transmission. *IEEE Trans. Veh. Technol.* **2020**, *69*, 2980–2992. [CrossRef]
5. Zambrano-Martinez, J.L.; Calafate, C.T.; Soler, D.; Lemus-Zúñiga, L.-G.; Cano, J.-C.; Manzoni, P.; Gayraud, T. A centralized route-management solution for autonomous vehicles in urban areas. *Electronics* **2019**, *8*, 722. [CrossRef]

6. Lee, I.; Lee, K. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Bus. Horiz.* **2015**, *58*, 431–440. [CrossRef]
7. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358. [CrossRef]
8. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
9. Roman, R.; Lopez, J.; Mambo, M. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. *Future Gener. Comput. Syst.* **2018**, *78*, 680–698. [CrossRef]
10. Li, Z.; Song, Y.; Bi, J. CADD: Connectivity-aware data dissemination using node forwarding capability estimation in partially connected VANETs. *Wirel. Netw.* **2017**, *25*, 379–398. [CrossRef]
11. Shi, W.; Dustdar, S. The promise of edge Ccomputing. *Computer* **2016**, *49*, 78–81. [CrossRef]
12. Chen, L.; Zhou, S.; Xu, J. Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. *IEEE/ACM Trans. Netw.* **2018**, *26*, 1619–1632. [CrossRef]
13. Kamoun, M.; Labidi, W.; Sarkiss, M. Joint resource allocation and offloading strategies in cloud enabled cellular networks. In Proceedings of the 2015 IEEE International Conference on Communications (ICC), London, UK, 8–12 June 2015; pp. 5529–5534.
14. Peng, E.; Li, Z. Optimal control-based computing task scheduling in software-defined vehicular edge networks, 2020. In Proceedings of the IEEE International Conference on Internet of Things and Intelligent Applications (ITIA), Zhenjiang, China, 27–29 November 2020.
15. Chalaemwongwan, N.; Kurutach, W. Mobile cloud computing: A survey and propose solution framework. In Proceedings of the 2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Chiang Mai, Thailand, 28 June–1 July 2016; pp. 1–4.
16. Dai, Y.; Xu, D.; Maharjan, S.; Zhang, Y. Joint load balancing and offloading in vehicular edge computing and networks. *IEEE Internet Things J.* **2019**, *6*, 4377–4387. [CrossRef]
17. Fan, W.; Liu, Y.; Tang, B.; Wu, F.; Zhang, H. TerminalBooster: Collaborative computation offloading and data caching via smart basestations. *IEEE Wirel. Commun. Lett.* **2016**, *5*, 612–615. [CrossRef]
18. Wang, M.; Jin, H.; Zhao, C.; Liang, N. Delay optimization of computation offloading in multi-hop ad hoc networks. In Proceedings of the 2017 IEEE International Conference on Communications Workshops (ICC Workshops), Paris, France, 21–25 May 2017; pp. 314–319.
19. Liu, L.; Chang, Z.; Guo, X.; Ristaniemi, T. Multi-objective optimization for computation offloading in mobile-edge computing. In Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC), Heraklion, Greece, 3–6 July 2017; pp. 832–837. [CrossRef]
20. Beraldi, R.; Mtibaa, A.; Alnuweiri, H. Cooperative load balancing scheme for edge computing resources. In Proceedings of the 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), Valencia, Spain, 8–11 May 2017; pp. 94–100.
21. Liu, L.; Chan, S.; Han, G.; Guizani, M.; Bandai, M. Performance modeling of representative load sharing schemes for clustered servers in multiaccess edge computing. *IEEE Internet Things J.* **2019**, *6*, 4880–4888. [CrossRef]
22. Luo, J.; Deng, X.; Zhang, H.; Qi, H. Ultra-low latency service provision in edge computing. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6.
23. Jian, C.; Chen, J.; Ping, J.; Zhang, M. An improved chaotic bat swarm scheduling learning model on edge computing. *IEEE Access* **2019**, *7*, 58602–58610. [CrossRef]
24. Yü, Y.; Prasanna, V.K. Energy-balanced task allocation for collaborative processing in wireless sensor networks. *Mob. Netw. Appl.* **2005**, *10*, 115–131. [CrossRef]
25. Tian, Y.; Ekici, E. Cross-layer collaborative in-network processing in multihop wireless sensor networks. *IEEE Trans. Mob. Comput.* **2007**, *6*, 297–310. [CrossRef]
26. Jin, Y.; Jin, J.; Gluhak, A.; Moessner, K.; Palaniswami, M. An intelligent task allocation scheme for multihop wireless networks. *IEEE Trans. Parallel Distrib. Syst.* **2011**, *23*, 444–451. [CrossRef]
27. Billet, B.; Issarny, V. From task graphs to concrete actions: A new task mapping algorithm for the future internet of things. In Proceedings of the 2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems, Philadelphia, PA, USA, 28–30 October 2014; pp. 470–478.
28. Colistra, G.; Pilloni, V.; Atzori, L. The problem of task allocation in the Internet of Things and the consensus-based approach. *Comput. Netw.* **2014**, *73*, 98–111. [CrossRef]
29. Chen, M.; Hao, Y. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 587–597. [CrossRef]
30. Subramanya, T.; Goratti, L.; Khan, S.N.; Kafetzakis, E.; Giannoulakis, I.; Riggio, R. SDEC: A platform for software defined mobile edge computing research and experimentation. In Proceedings of the 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, Germany, 6–8 November 2017; pp. 1–2.
31. Li, Z.; Xing, W.; Khamaiseh, S.; Xu, D. Detecting saturation attacks based on self-similarity of openflow traffic. *IEEE Trans. Netw. Serv. Manag.* **2019**, *17*, 607–621. [CrossRef]