electronics

MDPI

Article

An Efficient Task Scheduling Strategy Utilizing Mobile Edge Computing in Autonomous Driving Environment

Qi Liu [1,2], Zhigang Chen [1,2,*], Jia Wu [1,2,*], Yiqin Deng [1,2], Kanghuai Liu [1,2] and Leilei Wang [1,2]

1 Department of Software Engineering, School of Computer Science and Engineering, Central South University, Changsha 410075, China; llvbbo@163.com (Q.L.); dengyiqin@csu.edu.cn (Y.D.); liukanghuai@yeah.net (K.L.); wll_1234@163.com (L.W.)

2 "Mobile Health" Ministry of Education-China Mobile Joint Laboratory, Changsha 410083, China

* Correspondence: czg@csu.edu.cn (Z.C.); jiawu0510@csu.edu.cn (J.W.); Tel.: +86-133-8748-0797 (Z.C.); +86-182-7312-5752 (J.W.)

Received: 23 September 2019; Accepted: 22 October 2019; Published: 25 October 2019

Abstract: With the rapid development of various new types of services, autonomous driving has received extensive attention. Due to the dense traffic flow, the limited battery life and computing power of the vehicles, intelligent vehicles are unable to support some computationally intensive and urgent tasks. Autonomous driving imposes strict requirements on the response time of the task. Due to the strong computing power and proximity to the terminal of mobile edge computing (MEC) and the arrival of 5G, the task can be unloaded to MEC, and data can be exchanged in milliseconds, which can reduce the task execution time. However, the resources of the MEC server are still very limited. Therefore we proposed a scheduling algorithm that takes into account the special task of the autopilot. Tasks will select the appropriate edge cloud execution and schedule the execution sequence on the edge cloud by the scheduling algorithm. At the same time, we take the mobility of high-speed vehicles into consideration. The position of the vehicle can be obtained by the prediction algorithm, and the task results are returned to the vehicle by means of other edge clouds. The experimental results show that with the increase of the task amount, the algorithm can effectively schedule more tasks to be completed within the specified time, and in different time slots; it can also predict the location of the vehicle and return the result to the vehicle.

Keywords: autonomous driving; mobile edge computing; task scheduling; location prediction; autopilot task classification

1. Introduction

With the Internet of Things (IoT) and wireless technologies helping vehicles become smarter, human society has entered a new era of "all things connected, all things intelligent" [1,2]. Autopilot technology, which has provided us with better services, is the product of a deep integration of the automotive industry with the new generation of information technology, such as high-performance computing chips, artificial intelligence, and the Internet of Things. With the increase in vehicles, there are also some safety issues that need to be addressed. And as a key technology for future cars, it can significantly reduce traffic accidents caused by human factors and improve travel safety [3,4]. Self-driving vehicles need to obtain nearby environmental information from various sensor devices deployed around them, and then adjust themselves to the surroundings quickly, based on the results of the processing of the data, which poses a high demand for the time of task processing, such as vision-based target detection, radar data processing, sensing and end-to-end decision-making [5]. Since

the energy and computing power of vehicles are limited, they cannot meet the time requirements of these tasks. Therefore we consider that tasks can be unloaded to edge cloud (roadside lights, signs, etc.), which is a promising way to provide cloud computing capabilities within the radio access network in close proximity to mobile subscribers [6]. With the development of high-speed networks on wireless communication, especially the commercialization of 5G networks, vehicles can offload computing tasks to the edge cloud to greatly reduce the task processing time [7].

In recent years, communication among vehicles has also become a hot topic of research. Self-driving vehicles are about to begin, and a key milestone is coming with the introduction of short-range wireless communications among cars [8]. The past decade has seen the development of different wireless access technologies for vehicle-to-everything (V2X) communications, and an extensive set of related use cases have been drafted, each with its requirements [9]. However, visible light communication can be considered to complement other wireless communication technologies which could be overloaded in dense scenarios [10].

Mobile edge computing (MEC) is a new type of computing model proposed in recent years. MEC refers to an open platform that integrates network, computing, storage and application core capabilities on the side close to the source or data source [11]. Therefore, the delay-sensitive computing tasks can be offloaded to the edge cloud processing, enabling smart devices to obtain services from the edge computing servers (ECSs) of nearby base stations (BS), which greatly reduces the network delay [12]. This model is widely supported by academia and industry, and is considered to be one of the key technologies for next-generation networks. An increasing number of vehicles get access to nearby ECS, and ECS needs to collect and process a lot of data, thus causing network congestion. But the arrival of the 5G network further improves the practicality of MEC. Due to the high speed (transmission rate up to 10G/s), low latency (transmission delay is less than 1ms) of this 5G network, the data can be exchanged between the vehicle and the ECS within a few milliseconds [13–15]. Therefore, the combination of 5G and MEC can satisfy autonomous driving tasks owing to high-performance computing and low network latency.

An increasing number of autonomous-driving functions have been developed in the research related to autonomous driving, such as path planning, target detection, entertainment navigation, etc. Different tasks have different time requirements. These applications almost have the characteristics of the computationally intensive and delay sensitivity, while vehicle hardware is subject to limitations in terms of its computational capabilities and energy consumption. This poses a significant challenge for autonomous driving. In previous studies, the authors paid more attention to the processing of each task, rather than the scheduling between multiple tasks. In addition, the previous studies also assumed that the vehicle was stationary, without considering the impact of the vehicle's high-speed mobility and external environment on task processing.

When more and more time-constrained tasks are offloaded to the edge cloud for execution, the factors affecting the task response time are not the network delay or the distance of the offloaded transmissions of these 5G networks, but the task execution time required for processing the computationally intensive tasks. Moreover, in order to meet the time limit of each task, the task must be completed before the deadline, since the resources of the ECS are still very limited. The problems are converted into how to schedule tasks execution order to meet their requirement after they are offloaded to the edge cloud. For the car, they expect to get high-quality services; for the ECSs, they want to provide services for as many users as possible. The simplest solution is to execute the tasks in the order of their arrival, but this cannot fully utilize the resources of the ECS, and cannot satisfy the needs of more tasks. As a result, how to determine the order of the execution of these tasks, in order to make full use of the resources of the ECSs, and to make more tasks completed before the deadline, becomes a key issue [16].

According to the characteristics of the automatic driving task, this paper divided the task into three levels, and assigned corresponding priorities. At the same time, this paper has considered the influence of different external conditions on the vehicle task processing and its dynamical adjustment

of the task's deadline by integrating other external factors. Then, based on these characteristics of the task, we designed a new task scheduling algorithm, which enables ECSs to perform as many tasks as possible under the constraints of time and resources. It is called the priority urgency replacement strategy (PURS). When the task is processed, the task result needs to be returned to the vehicle. This paper has considered the particularity of autonomous driving tasks and the mobility of vehicles at high speeds [17]. Due to the high-speed mobility of the vehicle, the communication range of the base station is limited (the average coverage is about 200 square meters) [18,19], which means that the vehicle may not be within the communication range of the ECS of the task unloading.

Therefore we discussed two cases: If the vehicle is within the communication range of the current ECS, the result is directly transmitted back to the vehicle. If not, then we apply the Kalman filter algorithm to the next vehicle according to the historical trajectory of the vehicle. The position of the moment is predicted and positioned, and then the mission is transmitted to the vehicle by means of the other ECS. The main contributions of this paper are listed below:

- Due to the specificity of vehicle tasks, this paper assigns different priorities according to diverse types of tasks, and considers the impact of the external environment on the vehicle task execution time, and dynamically adjusts the mission's deadline.
- This paper proposes a new task scheduling scheme, which fully considers the particularity of vehicle tasks, and schedules the execution order of tasks by their property.
- Considering the characteristics of the high-speed mobility of vehicles, the position of the next time slot of the vehicle is predicted by analyzing the historical trajectory through Kalman filter algorithm.

2. Related Work

In recent years, with the rise of 5G and the Internet of Things, a number of people have shown a strong rise in the use of mobile technology to improve the performance of autonomous driving. There are a lot of researches on autonomous driving technologies, including detection radar, camera hardware, and other hardware design. At the same time, edge computing also provides enormous help for intelligent driving, like unloading, assisted driving, intelligent traffic, and so on [20,21].

2.1. Mobile Edge Computing

Edge computing has been studied as an extension of cloud computing that can provide real-time communication performance and strong computing power at the edge of the network. MEC is a supplement to cloud computing, not an alternative. Task scheduling is a traditional topic that involves transferring tasks to the external platform due to the limited computational power, storage and energy of the mobile device [20], which can improve computing efficiency, reduce task completion time, and utilize resources efficiently from other devices in the system [22,23]. Therefore, it has been extensively studied in wireless networks [24,25]

With the development of computing-intensive and time-sensitive applications, task scheduling is becoming a research hotspot for MEC. Mao et al. develop a dynamic task scheduling method for an MEC system with an energy harvest mobile device and an edge server [25]. In the context of the smart city, Deng et al. proposed an improved DIJ-ADMM task scheduling algorithm based on the ADMM algorithm, which effectively improved the task completion time and task unloading time [26].

Chen et al. analyze the issue of the unloading of computation intensive and data intensive tasks, dividing this optimization problem into two sub-problems: Task placement and resource allocation [27]. However, none of these studies take into account the particularity of the task, and cannot satisfy the needs of a particular scenario.

2.2. Autonomous Driving

The concept of connected vehicles was proposed in 1996, and many researchers have been working in this area to improve the safety and convenience of driving [28,29]. Then, many companies, research institutions and auto vendors showed great interest in autonomous driving [21,30]. Furthermore, some works have surveyed the current state-of-the-art planning and control algorithms with a particular emphasis on the urban setting [31,32]. However, wireless network bandwidth and real-time performance are often the bottlenecks of cloud computing for connected vehicles [28].

Integrating the IoT with mobile technology makes a variety of smart devices generate large amounts of data every day. For example, the Boeing 787 generates 5 GB of data per second and a self-driving car produces 4 TB of data per day [33]. As a result, data can be processed on the ECS with shorter response times, more efficient processing, and less network pressure.

With the arrival of 5G, the network transmission delay is no longer the most critical issue of self-driving. But how to deal with the data collected by various sensors in the shortest time is the most important thing [17,34].

Chowdhuri et al. consider the multiple behavior of autonomous driving modalities as distinct modes of an end-to-end autonomatic deep neural network by using the method of Multi-Modal Multi-Task Learning [35]. Blasinski et al. explain an approach to develop image system designs that meet the task requirements for autonomous vehicle applications [36]. Deyo et al. present an approach to the problem of Qualitative Autonomous Driving (QAD) using risk-bounded conditional planning [35]. These applications almost have the characteristics of being computationally intensive and sensitive to time delay, while vehicle hardware is limited in terms of their computational capabilities and energy consumption. This poses a significant challenge for autonomous driving.

Hou et al. [37] proposes a vehicular fog computing system which aggregates the underutilized communication and computation resources of individual vehicles to enhance the quality of services and applications. Neto et al. [38] present a cloud-based MEC offloading framework and designed an efficient computation offloading strategy to further reduce the latency and the transmission cost of the computation offloading.

There are many methods for vehicle trajectory prediction. Here we use the Kalman filter algorithm. Indeed, Simo et al. discuss connections of Gaussian process regression with Kalman filtering and present methods for converting spatiotemporal Gaussian process regression problems into infinite-dimensional state-space models [39]. In fact, this method is also used for crowd perception. Dardari et al. propose a combined Gaussian Process (GP)-State space method for crowd mapping whose complexity and memory requirements for field representation do not depend on the number of data measured [40].

3. System Model and Problem Formulation

In this section, we will introduce the system model and mathematically define the new metrics of the network. Then we give a definition of the proposed problem and provide a formulation for the problem.

In our system, we consider the task of autonomous driving particularity, such as the urgency of the task, the priority of the task, etc. Thus, the system is concerned with how to deal with the scheduling task and how to complete as many tasks as possible. Taking into account the mobility of vehicles on the road at high speed, we can predict the vehicle's position using the vehicle's historical trajectory. In the rest of this section, first, we have a preliminary analysis of the problem. Second, we design a dynamic adjustment of the task's deadline, and schedule the task to execute inside ECS. Third, we use Kalman filtering to predict the position of the vehicle and return the task results to the vehicle.

3.1. System Overview

As shown in Figure 1, self-driving vehicles use a variety of sensors to obtain surrounding environmental information. Then the result of these tasks are processed to determine the movement

of the vehicle, including path planning, overtaking acceleration, weather forecasting, navigation, entertainment, and so on. For these tasks' diversity and complexity, vehicles cannot process so many intensive tasks locally. Hence, the tasks needs to be offloaded to the edge device to meet the needs of autonomous driving. The vehicle can only communicate with the ECSs it covers.

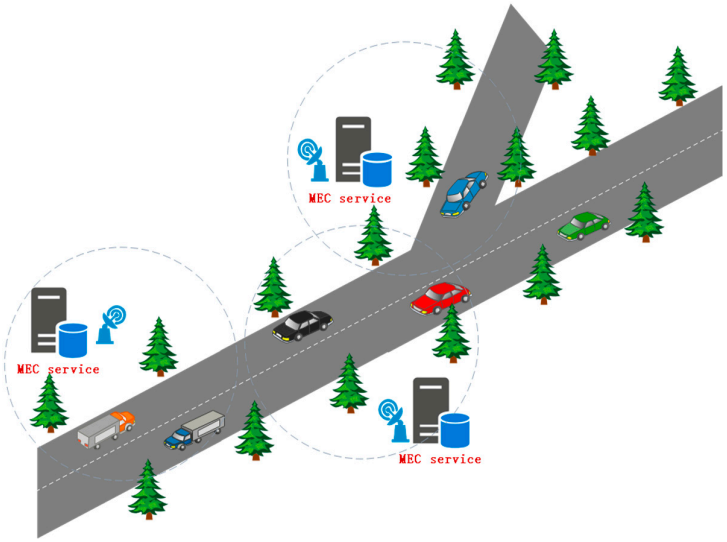

Figure 1. An illustration of the system model.

When the vehicle is ready to unload the task T_i to the surrounding ECSs, the vehicle will first send an unloading request to the nearest ECS it currently covers. This request includes all the information of T_i. Then ECS compares its remaining resources and requires time to complete the task so as to determine whether it can be executed here. If not, the PURS is used to select other ECSs to perform this task.

Due to the complexity of the road conditions, we divide the type of autopilot task into three levels: Common tasks (CTs), referring to tasks related to the user's entertainment, such as listening to songs, playing games, watching movies, online calls, browsing news. They will not affect the normal driving of the vehicle, and are only related to the user's experience. Therefore, such tasks have the lowest priority; important tasks (ITs), referring to tasks related to assisted driving and applications such as navigation, real-time traffic information, which assist drivers to drive safer. Without them, it would be inconvenient for the drivers; very important tasks (e.g., obstacle detection, lane following, path planning, etc. Which can affect vehicle speed, direction, road choice, etc.) (VITs). The assessment of the surrounding environment is a key part in autonomous driving and safety-related applications, such as path selection, acceleration, cruise control, braking . . . . Incorrect or overdue execution of such tasks will result in devastating consequences for vehicles and people. Hence VITs are given the highest priority. Compared to VITs, ITs can accept a longer waiting time. It is very important for the latter scheduling algorithm.

When the vehicle unloading task is initialized, the first step is to decide to which ECSs to offload the task. The 5G network enables the data transmission between the vehicle and ECSs to be completed in milliseconds. It is not the network transmission that affects the delay of the entire system at this time, but the execution time in the task processing. Therefore, the unloading time and return time of our task can be negligible. The time consumption of the whole system is only the waiting time of the task in the ECS and the execution of the task inside the ECS. When the task is unloaded to the ECS, and the scheduling algorithm is executed inside this ECS, we only consider the state inside the ECS at the current time (such as the number of tasks, waiting for queues), regardless of the impact of the subsequent time on the current task.

We define that, at initialization, the vehicle will send an unloading request to the nearest ECS. Then the ECS will determine whether the task can be executed in the current device according to our PURS algorithm. If yes, then it will be executed on the current ECS. If no, the vehicle will send an unloading request to all ECSs it covers, and select a suitable ECS to perform the task. Then the PURS schedules the execution order of the internal tasks of the ECS. In this way, we can minimize the time cost for the entire system while meeting our task requirements.

In our scheduling algorithm, the execution order of tasks in ECSs is the only point we focused on. If all the ECSs beside the vehicles cannot satisfy the task, some tasks can be migrated to other ECSs. The PURS not only satisfies the time-limited task of self-driving, but also satisfies the priority of different tasks. Therefore, we assume that the task execution in all edge devices conforms to our scheduling algorithm. When the task is processed, the result is returned to the vehicle. But in our model, the vehicles are moving at high speed. At this time, the vehicle may not be in the communication range of the ECS for task execution. Based on this, we use the Kalman filter algorithm to predict the location of the vehicle according to its historical trajectory. If the vehicle is not within the communication range of the ECS of the task execution, other ECSs need to be used to return the result to the vehicle. The meanings of the main symbols in this paper are summarized as Table 1.

Table 1. Lists of major used notations.

Notation	Description
S	the set of the ECSs in a given area
S_i	the ECS $i(S_i \in S)$
C_i	the computation ability of S_i
T	the set of n tasks
T_i	the task $i(T_i \in T)$
N_i	the computation workload of task T_i
O_i	the priority of task T_i
D_i	the deadline of task T_i
S_i	the set of external circumstances of T_i
ET_i^e	the execute time of task T_i
CT_i^w	the waiting time of task T_i
w_k	the weight of the external environment
$E_1^c, E_2^c, E_3^c, E_4^c$	current value of the external environment
$E_1^n, E_2^n, E_3^n, E_4^n$	standard value in the standard environment
D_i^r	adjusted deadline

3.2. System Model

In this section, we formulate the problem of task assignment among the ECSs. We define the existence of n ECSs in a given region, written as $S = \{S_1, S_2, \cdots S_n\}$. Moreover, for each $S_i \in S$, its computation ability is a constant C_i, which means that the data size it can process in unit time is C_i. We assume that there is only one ECS in each BS for simplicity.

Here, we define a set of n tasks as $T = \{T_1, T_2, \cdots T_n\}$.These tasks come from self-driving vehicles, and are performed on the ECSs in given areas. So we define the task as a four-tuple $T_i = \langle N_i, O_i, D_i, S_i \rangle$, where N_i is the computation workload of T_i and O_i is the priority of T_i, indicating the importance of the task, corresponding to the level of the task: CTs, ITs, VITs. Priority is not affected by the external environment, which is only related to the task itself. D_i is the last time that the task T_i needs to be completed. It reflects the urgency of task t. Since the vehicle is affected by many external environments, so D_i can be adjusted according to its own and environmental factors. In general, the deadline for VITs is earlier than that for CTs tasks. S_i is the extra case of task T_i, defined as a four-tuple $S_i = \langle WE_i, PE_i, VN_i, SP_i \rangle$. It represents the weather, the number of people in the car, the road congestion, and the speed of the vehicle. Note that we define a task T_i that can no longer be divided. Once T_i has selected a suitable ECS, it can only be executed on the ECS at the same time. In our model, the time cost CT_i of T_i consists of two parts: Execution time ${ET_i}^e$ and the waiting time ${CT_i}^w$.

As mentioned above, the computation ability of S_j is known to be C_j, and the size of T_i is N_i. Therefore, the calculation time ET_i^e of T_i on the S_j can be calculated:

$$ET_i^e = \frac{N_i}{C_j} \tag{1}$$

When T_i reaches an ECS, it will be added to a waiting queue. After the tasks in front of T_i are executed, T_i will be executed. We assume that there are n tasks in front of T_i in the waiting queue, then the waiting time CT_i^w can be calculated:

$$CT_i^w = \sum_{k=1}^{n} \frac{N_k}{C_j} + ET_i^e \tag{2}$$

where ET_i^e is the time it takes for the current task T_i to be executed on S_j. $\sum_{k=1}^{n} \frac{N_k}{C_j}$ is the task execution time in front of T_i.

In order to evaluate the performance of the PURS in the experiment, here we propose two task completion indicators: Task completion rate (TCR) and task time completion rate (TTCR). We assume that there are p tasks arriving at ECSs, and they are assigned to different ECSs for execution, and we note that there are q tasks are executed successfully. Therefore the task completion rate can be calculated through dividing the total of the tasks by the number of the successfully executed tasks.

$$TCR = \frac{q}{p} \tag{3}$$

The task time completion rate (TTCR) equals the total duration of the successfully executed task divided by the total task duration.

$$TTCR = \frac{\sum_{j=0}^{q} ET_j^e}{\sum_{i=0}^{p} ET_i^e} \tag{4}$$

where $\sum_{j=0}^{q} ET_j^e$ represents the total time spent in performing successful tasks and $\sum_{i=0}^{p} ET_i^e$ represents the total time spent performing these tasks.

4. PURS Scheduling Algorithm

Considering the characteristics of autonomous-driving tasks, the different urgency and the environment impacts, the aims of the scheduling algorithm is to solve the task assignment problem to make it so that more tasks can be executed before the deadline. In light of the time constraints and the priority of tasks, the scheduling algorithm determines which ECS the task should be assigned to.

When a new task reaches the ECS, current ECS will calculate task scheduling value (we will define this value in detail later) according to the priority and urgency of the task. Then, the scheduling algorithm determines whether the task can be executed on the current ECS. If yes, the task will be executed on the current ECS. If the current ECS cannot meet the constraint of the task, the vehicle sends the task unloading request to the nearby ECS. Each ECS executes the scheduling algorithm according to task scheduling value. The selected ECS satisfies the current task and the new task cannot affect the execution of other tasks on the current ECS after scheduling. If there are multiple ECSs to be chosen, we perform selection strategies based on different types of tasks: First execution strategy and last execution strategy. If all ECSs fail to execute the task, the PURS returns a message that no ECS can process the current task to the vehicle.

4.1. Adjust the Deadline of the Task Dynamically

Since vehicle travel is affected by many factors, such as weather conditions, the number of people in the car, road congestion, vehicle current speed, then the deadline of task processing is also affected accordingly. In order to dynamically adjust the tasks' deadline according to external factors, we give four main factors affecting the execution time of the vehicle task. ${E_k}^c$ denotes the current external situation, $k \in \{1, 2, 3, 4\}$ indicates the weather, number of people, number of vehicles and speed, respectively. D_i denotes the original deadline of task T_i. In order to satisfy the particularity of an autonomous driving task, the influence of external environment on the current vehicle task is illustrated by dynamically adjusting the deadline of the task. Here we can get the adjusted deadline:

$$D_i{}^r = D_i + \sum_{k=1}^{4} w_k({E_k}^c - {E_k}^n) \tag{5}$$

where w_k represents the effect weight of the external environment on the task deadline. ${E_k}^n$ indicates that the vehicle is in the general situation under the situation k. Then how to determine the value of ${E_k}^n$? Here we give the value of ${E_k}^n$ according to some common sense regulations or traffic regulations. Here are some examples. For the weather, we can divide the weather into three categories: Sunny (1), cloudy (2), rainy (3), and ${E_1}^n = 1$ denotes as sunny. For the number of people in the car, there are many types of vehicles. Each type of vehicle has a different number of passengers. Only the minimum value of 1 and the maximum value of 30 can be given here, so ${E_2}^c \in [1, 30]$ and ${E_2}^n$ is half of the rated passenger load of the vehicle. Road congestion can also be divided into three types: Sparse (1), normal (2), and congestion (3). ${E_3}^n = 2$ denotes as normal. For the given speed, due to the different speed limit standards given by each country, ${E_4}^n$ equals 60 km/h.

For the weight w, it is the factor of fine-tuning. You can adjust the size of the task deadline according to different situations dynamically. Therefore w can be positive or negative. For example, let us assume that the traffic of the current road is very congested, so we set ${E_3}^c = 3$ and $w = -0.5$, indicating that the current road condition is very poor, and the task must be executed 0.5 s ahead of time.

4.2. The Task PAD Calculation Method

In order to schedule the execution of tasks, we must give the task a scheduling value based upon the characteristics of the task. Therefore, we define the task's dispatch value as the pad. The PURS scheduling algorithm is based on the size of each task's pad. The idea of the PURS is to judge according to the urgency and the priority of the task. The smaller the pad, the earlier the task is executed, so here we define the value of the T_i pad as

$$pad_i = D_i{}^r - ET_i{}^e - O_i - AT_i \tag{6}$$

where $D_i{}^r$ is the adjusted deadline of T_i, $ET_i{}^e$ is the execution time of T_i, O_i is the priority for the task, AT_i is the current time for the task to reach the ECS. For example, we assume task T_i needs to be completed at 200 ms and itself must run for 100 ms and its priority is 10. Current time is 0, so the pad of T_i is: $pad_i = 200 - 100 - 0 - 10 = 90$ ms . Therefore, after obtaining the pad of each task, the task on the current ECS can be scheduled.

4.3. Scheduling of Tasks in ECSs

The core of the PURS is to dynamically adjust the execution order of the tasks on the ECSs according to the priority and urgency of the vehicle task specificity, so that the ECSs can perform more tasks and make full use of the resources of the ECSs. The entire scheduling process is as follows: Firstly, when S_c receives the task unloading request and T_i reaches S_c, S_c will calculate the pad of T_i, then adding T_i to the waiting queue Q_s^o of S_c. We define that there are l tasks waiting to be executed in the

waiting queue of S_c, then scheduling the task execution sequence on the basis of the size of the pad for each task. The smaller the pad for the task, the further forward in the waiting queue. When the scheduling algorithm is executed, a new waiting queue Q_s^n is formed. Whether T_i can be executed on the current ECS is judged by the new waiting queue.

Whether T_i can be executed on the current ECS not only depends on whether S_c can meet the time requirements for T_i, but also on whether T_i can affect the completion of the subsequent tasks when T_i is inserted into the waiting queue. Supposing that after the scheduling algorithm is executed, there is the m task in front of T_i and the n task after T_i. In other words, $m + n = l$. But T_i cannot affect the execution of tasks behind T_i in the new queue. Therefore, in order to meet the time requirement for T_i, the following equation must be satisfied.

$$CT_i^e + CT_i^w \leq D_i^r \tag{7}$$

where $CT_i^w = \sum_{k=1}^{m} \frac{N_k}{C_j} + ET_i^e$ according to Equation (2)

Besides, as we presuppose that the scheduling of new tasks will not impact the running of previous tasks, so those n tasks behind T_i must satisfy the following equation

$$CT_j^{prev} + CT_j^e + CT_i^e \leq D_j^r, \forall j \geq n \tag{8}$$

where CT_j^{prev} refers to the waiting time of the task T_j before T_i is added into the waiting queue Q_s^n. As a result, task T_i can run on ECS S_c if and only if the above two formulas are met.

The PURS algorithm is a preemption algorithm. Thus we provide the PURS preemption mechanism: When the current task is completed, the task of the pad becomes zero. We only focus on the order of the execution of the task, while not caring about how to perform the task inside. The advantages of the algorithm is that we do not worry about the arrival order of these tasks. The algorithm takes into account the urgency and priority of the task.

Hypothesizing that there are currently three tasks T_1, T_2, T_3 reaching the ECS S_1, we assume that the external environment is good, that is, it has no effect on the deadline of the task. The computing power of the ECSs is the unit computing power. $T_1 = \langle 100, 100, 200, 0 \rangle$, $T_2 = \langle 150, 100, 400, 0 \rangle$, $T_3 = \langle 100, 50, 700, 0 \rangle$. According to $pad_i = D_i^r - ET_i^e - O_i - AT_i$, we can know in the initial state: $pad_1 = 0$, $pad_2 = 150$, $pad_3 = 550$, so T_1 is executed first. During T_1 execution, no task pad value is 0, so there is no task preemption. T_1 is completed normally. Then $pad_2 = 50$, $pad_3 = 450$, and now T_2 is executed. During T_2 execution, no task pad value is 0, and T_2 is fulfilled normally. At last T_3 is executed, which just meets the requirements. The task execution strategy algorithm shows in Algorithm 1. The details are shown in Figure 2.

Time	pad(T_1)	pad(T_2)	pad(T_3)
0	0	150	550
100		50	450
250			300

Figure 2. Task execution sequence diagram.

Algorithm 1 ECS Internal Task Execution Strategy

Input: task T_i, ECS S_j
Output: if T_i can be executed on S_j
T_i denotes a newly-arrived task
S_j denotes the ECS performing T_i
Q_s^n denotes the task waiting queue for ECS S_j
Begin
Calculate the pad of T_i according to Formula (6).
Insert task T_i into the wait queue of S_j^w
For all $T_k \in Q_s^n$ **do**
    If ($pad_i \leq pad_k$) **then**
        exchange T_i and T_k location
    End if
End for
If (whether T_i, T_k satisfied the Formulas (6) and (7)) **then**
    Return S_j can execute T_i
Else
    Return S_j cannot execute T_i
End if

4.4. Choice of ECSs and Task Migration

When a new task unloading a request reaches an ECS closest to the vehicle, the ECS calculates the pad of the task and then determines whether the task can be executed on the current ECS, based on the pad value of each task. If the task can be executed on the current ECS, that is the best choice. If the current ECS cloud cannot be executed, the vehicle sends a task unloading request to the ECSs around it. Based on the pad value of each task, the PURS is performed to determine whether there is an ECS that can meet the task's execution. We define the ECSs collection that satisfies T_i as S^s. So S^s can be divided into two categories according to the position of T_i in the waiting queue: First execution strategy (FES) and last execution strategy (LES). According to the position of T_i in the waiting queue, S^s is divided into two parts: S_e^s and S_l^s. S_e^s indicates that T_i is in the first half of the waiting queue, and S_l^s indicates that T_i is in the second half of the waiting queue. Hence, we can choose the appropriate ECSs according to the type of T_i.

First execution strategy: Because of the particularity of the task, some tasks need to be executed as soon as possible, such as VITs. So the strategy is to meet this kind of high priority task and to meet the priority of short tasks, which improves the completion rate of these tasks. Prior execution of short tasks has less impact on the wait time. It is great to improve the efficiency of the ECSs.

Last execution strategy: For some CTs tasks, the latency requirements are not particularly high, as long as they are completed within the specified time. Therefore, we can put this part of the task behind the waiting queue to save preempting resources for highly urgent tasks. As for some long tasks, if such tasks are executed first, it will greatly increase the waiting time for latter tasks in the waiting queue and reduce the system efficiency. So the final execution strategy can be preferred for both types of tasks.

If the PURS is executed, S^s cannot fulfill task execution. At this time, we need to migrate some tasks from the current ECS to other ECSs for execution, so that each task can be finished. Considering of the different task sizes, we assume that the task selected for migration is a short task.

When T_i reaches S_t, there are k tasks waiting to be executed currently. Due to insufficient resources, some tasks need to be scheduled to other ECSs to satisfy T_i. Then, assuming that T_j will be migrated to other ECSs, it needs to satisfy the conditional Formulae (7) and (8) above. That is to ensure that the T_j can be performed normally after being scheduled to another ECS. At the same time, we presume the migratory task is shorter than the current task: ${CT_j}^e < {CT_i}^e$. The ECS selection algorithm shows in Algorithm 2.

Algorithm 2 ECS Selection

Input: task T_i, ECS S
Output: S_e
T_i denotes a newly-arrived task
S denotes the collection of clouds near the edge of the vehicle
S_c denotes the ECS closest to the T_i, and S^a denotes the collection of ECS can execute T_i
Begin
If (according to algorithm 1, judge whether T_i can be executed on S_c) **then**
    Return S_c
Else
    For all $S_j \in S$ **do**
        If (according to algorithm 1, judge whether T_i can be executed on S_j) **then**
            Add S_j to S^a
        End if
    End for
End if
If (S^a is empty) **then**
    migrate some task to their corresponding ECS
    If (according to algorithm 1, judge whether can T_i be executed on S_c)
        Return S_c
    Else
        Return no suitable ECSs to meet the requirements of T_i
    End if
Else
    according to the size and urgency of T_i choose EES or LES to select most suitable S_m
    Return S_m
End if

4.5. Predict the Location of the Vehicle and Return the Result to the Vehicle

Due to the high-speed movement of the vehicle, when the vehicle unloads the task onto the ECS to execute, how can the finished results be returned to the vehicle? That means, how to determine whether the current vehicle is still in the communication range of the ECSs. If the vehicle is still in the communication range of the current ECS, the ECS can directly return the results to the vehicle. If not, the current task processes need to return the results to the vehicle with the help of another ECS. Then how can the position of the vehicle during the task execution be determined? Here we use the Kalman filtering to make a dynamic trajectory prediction of the vehicle according to the vehicle's previous trajectory.

Each vehicle has GPS for positioning, and is moving at high speed. In order to more accurately predict the position of the vehicle after the ECS has finished the task, we establish a coordinated system. Assuming that the vehicle is moving in the x axis in a two-dimensional plane, the direction perpendicular to the road is the y axis. The vehicle stores the position information at different times during its moving motion.

The position points vector set $Tra = \{Tr_1, Tr_2, \cdots Tr_n\}$. $Tr_i = (\alpha_i, \beta_i)$ can be composed according to the temporally-ordered form, where α_i, β_i are the projection vectors of Tr_i on the x axis and y axis, respectively.

The equation of system state is expressed as:

$$X(k) = \Phi(k)X(k-1) + \Gamma(k)U(k) + W(k) \tag{9}$$

where $X(k)$ is an n dimensional state vector, indicating the state of system at time k. $\Phi(k)$ is the state transition matrix of the system, describing the parameter matrix of the state of system transitioning from time $k-1$ to time k. $\Gamma(k)$ is the input control model acting on the controller vector $U(k)$. $U(k)$ is

the control input vector at time k. $W(k)$ is the system process noise used to describe noise or error that shifts from one state to others.

The dynamic system observation equation is:

$$Z(k) = H(k)X(k) + V(k) \tag{10}$$

where $Z(k)$ is a dimensional observation vector, indicating the state in which the system is measured at time k. $H(k)$ is an observation model, which is the mapping between real state space and observation space. $V(k)$ is an observed noise sequence that describes the noise or error of a dynamic system moving from one state to another. $H(k)$ is the $m \times n$ dimensional observation matrix. The observed noise $V(k)$ is a zero-mean white noise sequence. In general, the noise vector $W(k)$ of the assumed system and the noise vector $V(k)$ of the dynamic system measurement are both Gaussian white noise with a mean value of zero.

It is assumed that the initial values of the states are not related to $X(0)$ of the states are not related to $W(k)$, $V(k)$. That is, $E(W(k)V^{(T)}(k)) = 0$ The dynamic system measurement data vector $Z(1), Z(2), Z(3), \cdots Z(k)$ are used to find the system state vector $X(i)$ component for least squares estimation. According to the different relationship between i and n, the Kalman filter can be divided into three cases: Filtering, smoothing and prediction.

The core of the Kalman filter algorithm is to use the recursive algorithm to achieve the estimation model of the optimal state estimation, and use the estimated value of the previous moment and the observation value of the current moment to update the estimation of the current state variable. Based on the previous k observations, the optimal state estimation at time k is obtained. There are two different update processes in the process of the stochastic linear discrete Kalman filter, which are the time update process and the observation update process. The time update process predicts the state at the current time based on the optimal state estimation at the previous moment. At the same time, the covariance of the current prediction state is updated, and the time update is

$$X(k+1,k) = A(k)X(k,k) \tag{11}$$

$$Z(k+1,k) = H(k)X(k+1,k) \tag{12}$$

After predicting the trajectory point, it is necessary to use the observation value to linearly fit the position of the optimal estimated trajectory point. That is to estimate the optimal estimation point by observing the update equation based on the observed value and the predicted value and the update observation equation expression:

$$B(k+1) = Z(k+1) - Z(k+1,k) \tag{13}$$

$$X(k+1,k+1) = X(k+1,k) + K(k+1)B(k+1) \tag{14}$$

In the prediction process, the formula of the gain matrix K can be obtained from the initial state estimation values calculated by the above filtering process is

$$K(k) = A(k)P(k,k-1)H^{(T)}(k)[H(k)P(k,k-1)H^{(T)}(k) - R(k)]^{(-1)} \tag{15}$$

After obtaining the gain matrix K, according to the formula of the optimal predictive estimation equation:

$$X(k+1,k) = A(k)X(k,k-1) + K(k)[Z(k) - H(k)X(k,k-1)] \tag{16}$$

the predicted value of next time $X(k+1,k)$ can be calculated, and meanwhile we update the estimated error variance matrix $P(k+1,k)$:

$$P(k+1,k) = A(k)P(k,k-1)A^{(T)}(k) - A(k)P(k,k-1)H^{(T)}(k)[H(k)P(k,k-1)H^{(T)}(k) + R(k)]^{(-1)}H(k)P(k,k-1)A^{(T)}(k) + T(k)Q(k)T^{(T)}(k) \quad (17)$$

According to the above formula, the optimal prediction value at the next moment is obtained, and the single-step prediction process is completed. If the k step(s) is to be predicted, predicting k times iteratively.

After the current position of the vehicle is obtained, the result can be returned to the vehicle. The vehicle trajectory prediction algorithm shows in Algorithm 3.

Algorithm 3 Vehicle Trajectory Prediction

Input: vehicle trajectory data set $Tra = \{Tr_1, Tr_2, \cdots Tr_n\}$
Output: vehicle track point position
1. **Begin**
2. B = preprocess(Tra);
3. init();
4. state = currentState (B)
5. **For** $i = 1$ **to** k **do**
6. l' = KalmanPredict (B)
7. $error_i = \frac{\sum_{i=1}^{k} \sqrt{(x_i' - x_i)^{(2)} + (y_i' - y_i)^{(2)}}}{k}$
8. **End for**
9. $E_i = \frac{(\sum_{i=1}^{k} error_i)}{k}$
10. **Return** l'

5. Experiments and Results

In this section, we evaluate the proposed algorithms and task scheduling performance in different situations, as well as the accuracy of vehicle location prediction in different states.

5.1. Simulation Settings

The simulations are developed by MATLAB. We build a simulation platform for multitasking real-time systems with its own Simulink software package and the Turetime-1.5 toolbox to build a simulation platform for multitasking real-time systems. On the basis of the system model, in order to provide versatility, we set up a large number of automatic driving vehicles on the highway, setting the distance between 5G base stations to 200 m.

New tasks with different degrees of urgency are constantly generated in the process of driving vehicles, and the request for unloading tasks is first sent to the ECS in the latest BS. When the ECS receives various tasks, it will follow the PURS scheduling algorithm proposed above. After execution, the location of the vehicle is predicted by the Kalman filtering algorithm, and the results are transmitted back to the vehicle.

The parameters that need to be set for the experiment are: Calculating the workload N_i of the task T_i, task priority O_i, original deadline D_i, external system factor S_i, computing power C_j for each ESC. Now we can calculate the execution time CT_i^c of T_i based on the previous formula. For the external situation, we have already set the parameters above, and the specific experimental parameters on simulation such as the error threshold and noise variance of the vehicle trajectory prediction are shown in the following Table 2.

Table 2. Application parameter setting in the simulation.

Parameters	Default Values
C_i	Constant 1
N_i	Rand(0.5sec,4sec)
O_i	Rand(0.5,1.5)
D_i	Rand(2sec,8sec)
w_k	Rand(−0.5,0.5)
$E_1^c, E_2^c, E_3^c, E_4^c$	Rand(1,3),Rand(1,30),Rand(1,3),Rand(30,120)
$E_1^n, E_2^n, E_3^n, E_4^n$	1, half of E_2^c,2,30
Moving object trajectory data set	GeoLife, T-Drive
Prediction error threshold	20 m
Observation noise covariance	[10, 0; 0, 10] [1, 0, 0, 0; 0, 10, 0, 0; 0, 010, 0; 0, 0, 0, 10]
System noise covariance	[10, 0, 0, 0; 0, 10, 0, 0; 0, 0, 10, 0; 0, 0, 0, 10] [100, 0, 0, 0; 0, 100, 0, 0; 0, 0, 100, 0; 0, 0, 0, 100]

Note that these values of tasks are arbitrarily set because they vary among different autonomous-driving applications. Various values are investigated, and are within a reasonable range.

5.2. Simulation Results

In Figures 3 and 4, in order to verify the advantages of our PURS scheduling algorithm, we compare the direct execution strategy (DE) in the order of reaching the ECS. In the first set of simulations, we set 15 ECSs in the system, 50 tasks in each group, and therefore 750 tasks in total. The ratio of the three priority tasks is CTs:ITs:VITs = 3:1:1. Above all, the two metrics CR and CWCR are compared. As shown in the figure, when the ESCs task is less than 10, the ECSs resources are sufficient to satisfy the task. Hence, the TCR and TTCR of the direct execution strategy and PURS are both 100%. When the number of tasks is greater than 10, on account of the arrival of some high-priority tasks in the waiting queue, sequential execution cannot meet the time constraint requirements of these high-priority tasks. After being scheduled by the PURS algorithm, tasks with high priority and high urgency can be effectively performed to satisfy most tasks. However, as the number of tasks increases, the computing resources of ECSs tend to be exhausted, and both TCR and TTCR will decrease. Yet, the execution order of the task scheduled by PURS is obviously better than the direct execution strategy. At this time, individual ESCs cannot satisfactorily perform tasks through the scheduling algorithms. Therefore, other ESCs are needed to assist in the migration of some short tasks to other ECSs.

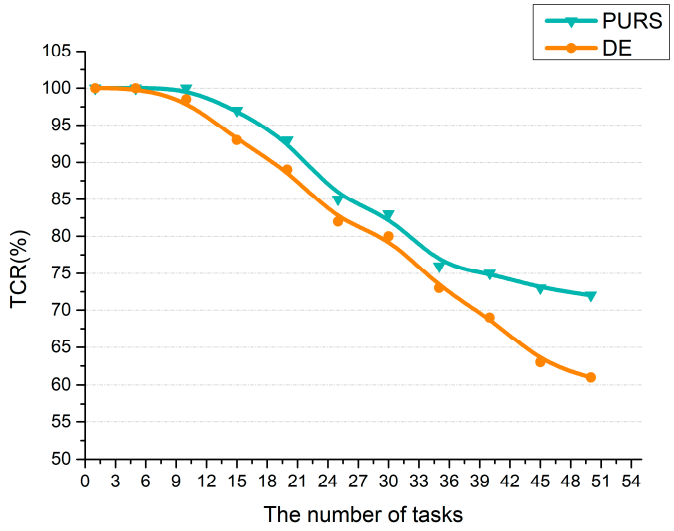

Figure 3. The task completion rate (TCR) of two algorithms.

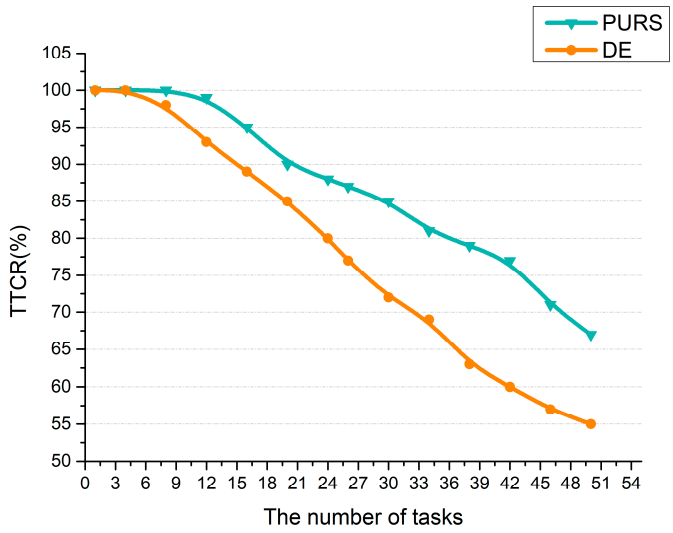

Figure 4. The task time completion rate (TTCR) of two algorithms.

In order to compare the scheduling capabilities of the PURS for different priority tasks, we reset the task priority ratios to CTs:Its:VITs = 3:1:1, CTs:Its:VITs = 1:3:1 and CTs:Its:VITs = 1:1:3. By comparing the size of TCR under different priority task ratios, the PURS is in line with the requirements of automatic driving task processing and perform better when scheduling multi-tasks with higher priority. As is shown in Figure 5, high-priority tasks are produced most by self-driving vehicles during driving. However, as the number of tasks increases, a single ECS cannot satisfy a large number of simultaneous tasks. Therefore, multiple ECSs need to cooperate directly.

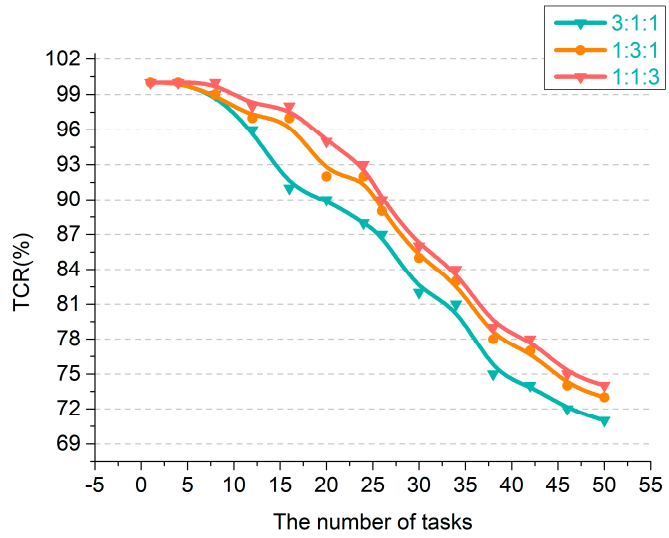

Figure 5. The TCR of different proportions of tasks.

When a single ECS cannot complete a growing number of tasks, we can offload some short tasks of the current ECS to the nearby ECS, and set 10 ECSs in the system, 50 tasks in each group, and thus, 500 tasks in total. The ratio of the three priority tasks is CTs:Its:VITs = 3:1:1. We directly compare the TCR of a single ECS and multiple ECSs under multitasking. As can be seen in Figure 6, the coordinated scheduling between multiple ECSs significantly increases the completion rate of the task, which is close to 90%. In practice, more ECSs can be deployed on the roads with more vehicles based on actual conditions, further improving the task completion rate.

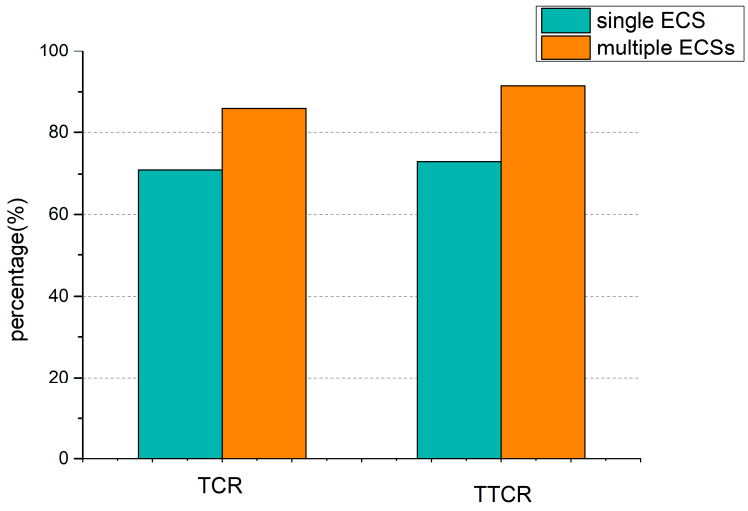

Figure 6. TCR of single edge computing servers (ECS) and multiple ECSs.

In the practical applications, completing more tasks of VITs wins more gains than the losses caused by proper extensions of deadlines of CTs and ITs. Therefore, task urgency classification and task deadline determination make contributions to the completion of both more tasks and more urgent tasks.

Figures 7 and 8 focus on the performance of track prediction algorithms. When the task is completed, our Kalman filter is used to predict the position of the vehicle. We obtain the prediction accuracy and prediction time under different data sets through experiments. When the number of predicted track segments increases, the accuracy rate also rises slowly.

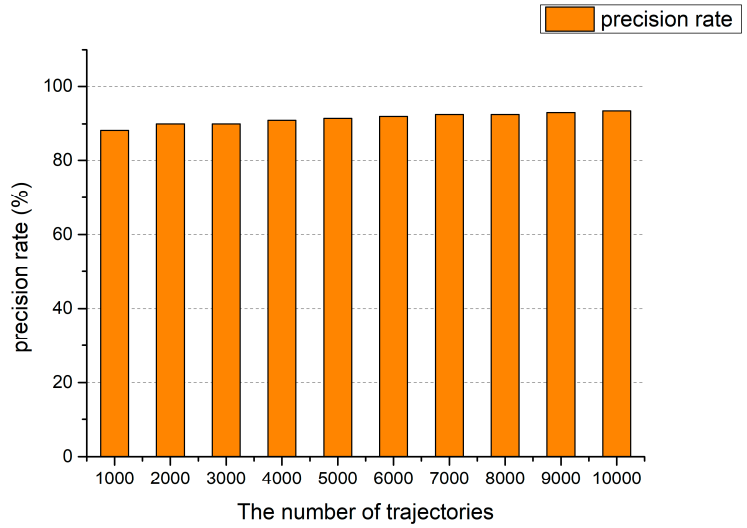

Figure 7. Prediction accuracy of different numbers of historical trajectories.

However, the prediction time increases accordingly. Because the communication range of ECSs is very large, the impact of small prediction errors on the whole system is so tiny as to be negligible. Therefore, fewer historical trajectories can be selected to predict the position of the vehicle, and the extra overhead of the system is also reduced.

For different data sets, we can adjust the range of parameters appropriately, such as the weight of external factors, depending on the characteristics of different tasks. In addition, the weights of these factors are based on common sense, so they can be learned through machine learning based on enough autonomous-driving data, and in this way our method can adapt to more autopilot scenes and other tasks scheduling situations. Through the experiment, we can conclude that our scheduling

algorithm can effectively improve the utilization of the edge cloud and reduce the processing time of the self-driving task. This provides an effective solution for handling the self-driving tasks.

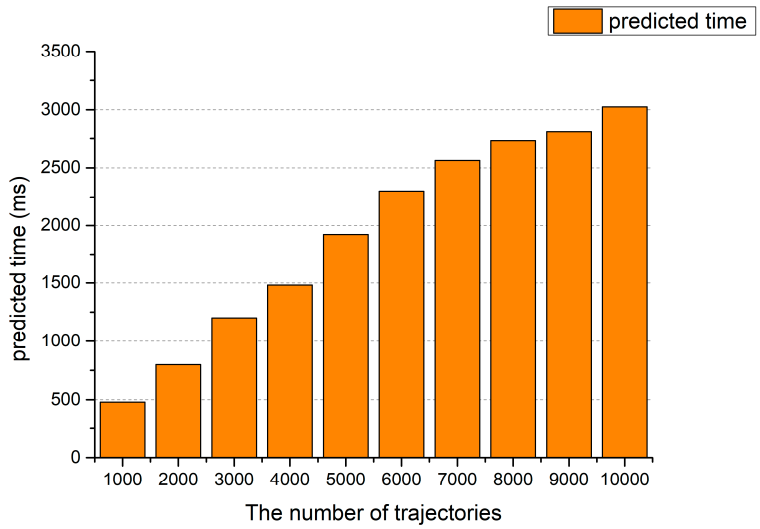

Figure 8. Time consumption of different numbers of historical trajectories.

6. Conclusions

Considering the particularity of autonomous driving tasks and the arrival of 5G networks, energy consumption and time delay are not the biggest problems. But rather, how to schedule tasks executed on ECSs to satisfy more tasks' requirements becomes a key issue. Since self-driving vehicles are affected by plenty of external environments, we propose to adjust the tasks deadline dynamically. Then on the basis of the priority and urgency, the task execution order is scheduled to fulfill more tasks, including the selection of ECSs and the scheduling of the internal tasks of ECSs.

The experimental results prove that, compared with the first-come first-served direct execution strategy, PURS has a significant improvement in TCR and TTCR. The cooperation between multiple ECSs further improves the task completion rate of the entire system. Therefore, the PURS algorithm can solve the problem of the execution order of intelligent driving intensive tasks in the edge cloud. On this basis, we also consider the high-speed vehicles' movement, in the case of which the cars may not be within the communication range of the ECS of task execution. Therefore, the vehicle position prediction based on a Kalman filtering algorithm is proposed, and the experimental results also prove the feasibility of the proposed algorithm, which is applicable to the actual scene. Much work is needed in the future research. The next work is to build a real scene and improve our algorithm by simulating more problems encountered in the actual situation.

Author Contributions: Conceptualization, Q.L. and J.W.; Methodology, Q.L.; Software, Q.L.; Validation, Y.D., K.L. and L.W.; Formal Analysis, Z.C.; Investigation, Z.C.; Resources, Q.L.; Data Curation, K.L.; Writing—Original Draft Preparation, Q.L.; Writing—Review & Editing, Q.L.; Visualization, J.W.; Supervision, J.W.; Project Administration, Q.L.; Funding Acquisition, Z.C.

Funding: This work was supported by National Natural Science Foundation of China (No. 61672540), Hunan Provincial Natural Science Foundation of China (No. 2019JJ50802), and Graduate Research and Innovation Project of Hunan (No. 2019zzts068). Furthermore, it was partially supported by the Major Program of National Natural Science Foundation of China (No. 71633006) and "Mobile Health" Ministry of Education—China Mobile Joint Laboratory.

Acknowledgments: This work was supported partially by "Mobile Health" Ministry of Education—China Mobile Joint Laboratory.

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

References

1. Farhan, L.; Shukur, S.T.; Alissa, A.E.; Alrweg, M.; Kharel, R. A survey on the challenges and opportunities of the Internet of Things (IoT). In Proceedings of the 2017 Eleventh International Conference on Sensing Technology (ICST), Sydney, Australia, 4 December 2017.
2. Guo, L.; Chen, Z.; Zhang, D.; Liu, J. Sustainability in Body Sensor Networks with Transmission Scheduling and Energy Harvesting. *IEEE Internet Things J.* **2019**. [CrossRef]
3. Anderson, J.M.; Nidhi, K.; Stanley, K.D.; Sorensen, P.; Samaras, C.; Oluwatola, O.A. Constantine. In *Autonomous Vehicle Technology*; RAND Corporation: Santa Monica, CA, USA, 2014.
4. WU, J.; CHEN, Z.; ZHAO, M. "Community Recombination and Duplication Node Traverse Algorithm in Opportunistic Social Networks", Peer-to-Peer Networking and Applications. 2019. Available online: http://faculty.csu.edu.cn/jiawu/zh_CN/lwcg/83061/content/22911.htm (accessed on 23 September 2019).
5. Giordano, G.; Segata, M.; Blanchini, F.; Lo Cigno, R. A joint network/control design for cooperative automatic driving. In Proceedings of the 2017 IEEE Vehicular Networking Conference (VNC), Torino, Italy, 27–29 November 2017.
6. Kiani, A.; Ansari, N. Edge Computing Aware NOMA for 5G Networks. *IEEE Internet Things J.* **2018**, *1*, 1. [CrossRef]
7. Mao, Y.; Zhang, J.; Letaief, K.B. Joint Task Offloading Scheduling and Transmit Power Allocation for Mobile-Edge Computing Systems. In Proceedings of the 2017 IEEE Wireless Communications and Networking Conference (WCNC), San Francisco, CA, USA, 19 March 2017.
8. Bazzi, A.; Cecchini, G.; Menarini, M.; Masini, B.M.; Zanella, A. Survey and Perspectives of Vehicular Wi-Fi versus Sidelink Cellular-V2X in the 5G Era. *Future Internet* **2019**, *11*, 122. [CrossRef]
9. Masini, B.; Bazzi, A.; Zanella, A. A survey on the roadmap to mandate on board connectivity and enable V2V-based vehicular sensor networks. *Sensors* **2018**, *18*, 2207. [CrossRef]
10. Masini, B.; Bazzi, A.; Zanella, A. Vehicular visible light networks for urban mobile crowd sensing. *Sensors* **2018**, *18*, 1177. [CrossRef]
11. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
12. You, C.; Huang, K.; Chae, H.; Kim, B.H. Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading (Extended Version). *IEEE Trans. Wirel. Commun.* **2017**, *16*, 1397–1411. [CrossRef]
13. Chen, Z.; Guo, L.; Zhang, D.; Chen, X. Energy and channel transmission management algorithm for resource harvesting body area networks. *Int. J. Distrib. Sens. Netw.* **2018**, *14*, 1–14. [CrossRef]
14. Li, S.; Li, D.X.; Zhao, S. 5G Internet of Things: A Survey. *J. Ind. Inf. Integr.* **2018**. [CrossRef]
15. Song, H.; Fang, X.; Yan, L. Handover Scheme for 5G C/U Plane Split Heterogeneous Network in High-Speed Railway. *IEEE Trans. Veh. Technol.* **2014**, *63*, 4633–4646. [CrossRef]
16. Abbas, N.; Zhang, Y.; Taherkordi, A.; Skeie, T. Mobile Edge Computing: A Survey. *IEEE Internet Things J.* **2017**, *99*, 1. [CrossRef]
17. Zhang, S.; Deng, W.; Zhao, Q.; Sun, H.; Litkouhi, B. Dynamic Trajectory Planning for Vehicle Autonomous Driving. In Proceedings of the IEEE International Conference on Systems, Manchester, UK, 13–16 October 2013.
18. PConline. Available online: https://product.pconline.com.cn/itbk/top/1903/12426316.html (accessed on 23 September 2019).
19. Dai, H.; Zeng, X.; Yu, Z.; Wang, T.T. A scheduling algorithm for autonomous driving tasks on mobile edge computing servers. *J. Syst. Archit.* **2019**, *14*, 23. [CrossRef]
20. Khan, A.U.R.; Othman, M.; Khan, A.N.; Shuja, J. Computation Offloading Cost Estimation in Mobile Cloud Application Models. *Wirel. Pers. Commun.* **2017**, *97*, 4897–4920. [CrossRef]
21. Ford Will Have a Fully Autonomous Vehicle in Operation by 2021. Available online: https://corporate.ford.com/innovation/autonomous-2021.html (accessed on 12 September 2019).
22. Yuan, W.; Ping, Q.L.; Haowei, M.; Yang, X.W.; Zhou, H.B.; Tan, X.Q. Secrecy-Driven Resource Management for Vehicular Computation Offloading Networks. *IEEE Netw.* **2018**, *32*, 84–91.
23. Yao, X.; Zhang, R.; Zhang, Y.; Lin, Y. Verifiable social data outsourcing. In Proceedings of the IEEE INFOCOM 2017–IEEE Conference on Computer Communications, Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.

24. Deng, Y.; Chen, Z.; Zhang, D.; Zhao, M. Workload scheduling toward worst-case delay and optimal utility for single-hop Fog-IoT architecture. *IET Commun.* **2018**, *12*, 2164–2173. [CrossRef]
25. Hao, Y.; Chen, M.; Hu, L.; Hossain, M.S.; Ghoneim, A. Energy efficient task caching and offloading for mobile edge computing. *IEEE Access* **2018**, *6*, 11365–11373. [CrossRef]
26. Deng, Y.; Chen, Z.; Yao, X.; Hassan, S.; Wu, J. Task scheduling for smart city applications based on multi-server mobile edge computing. *IEEE Access* **2019**, *7*, 14410–14421. [CrossRef]
27. 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]
28. Lu, N.; Cheng, N.; Zhang, N.; Shen, X.M.; Mark, J.W. Connected Vehicles: Solutions and Challenges. *IEEE Internet Things J.* **2014**, *1*, 289–299. [CrossRef]
29. Xia, C.; Jin, X.; Kong, L.; Xu, C.; Zeng, P. Lane scheduling around crossroads for edge computing based autonomous driving. *J. Syst. Archit.* **2019**, *95*, 1–8. [CrossRef]
30. Zhou, Y.; Tuzel, O. Voxelnet: End-to-end learning for point cloud based 3d object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, South Lake City, UT, USA, 2018; pp. 4490–4499.
31. Schoettle, B.; Sivak, M. *A Survey of Public Opinion about Autonomous and Self-Driving Vehicles in the US, the UK, and Australia*; University of Michigan: Ann Arbor, MI, USA, 2014.
32. Michels, J.; Saxena, A.; Ng, A.Y. High speed obstacle avoidance using monocular vision and reinforcement learning. In Proceedings of the 22nd International Conference on Machine Learning, Bonn, Germany, 7–11 August 2005; pp. 593–600.
33. Boeing 787s to Create Half a Terabyte of Data per Flight, Says Virgin Atlantic. Available online: https://datafloq.com/read/self-driving-carscreate-2-petabytes-data-annually/172 (accessed on 7 December 2016).
34. Haenggi, M.; Andrews, J.G.; Baccelli, F.; Dousse, O.; Franceschetti, M. Stochastic geometry and random graphs for the analysis and design of wireless networks. *IEEE J. Sel. Areas Commun.* **2009**, *27*, 1029–1046. [CrossRef]
35. Chowdhuri, S.; Pankaj, T.; Zipser, K. MultiNet: Multi-Modal Multi-Task Learning for Autonomous Driving. In Proceedings of the 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), Hawaii, HI, USA, 7 January 2019.
36. Blasinski, H.; Farrell, J.; Lian, T.; Liu, Z.; Wandell, B. Optimizing Image Acquisition Systems for Autonomous Driving. *Electron. Imaging* **2018**, *5*, 1–7. [CrossRef]
37. Hou, X.; Li, Y.; Chen, M.; Wu, D.; Jin, D.; Chen, S. Vehicular Fog Computing: A Viewpoint of Vehicles as the Infrastructures. *IEEE Trans. Veh. Technol.* **2016**, *65*, 3860–3873. [CrossRef]
38. Jose, L.D.; Neto, Y.S.Y.; Macedo, D.F.; Nogueira, J.M.S.; Langar, R.; Secci, S. ULOOF: A User Level Online Offloading Framework for Mobile Edge Computing. *IEEE Trans. Mob. Comput.* **2018**, *17*, 2660–2674.
39. Sarkka, S.; Solin, A.; Hartikainen, J. Spatio-temporal learning via infinite-dimensional Bayesian filtering and smoothing. *IEEE Signal Process. Mag.* **2013**, *30*, 51–61. [CrossRef]
40. Dardari, D.; Pasolini, G.; Zabini, F. An efficient method for physical fields mapping through crowdsensing. *Pervasive Mob. Comput.* **2018**, *48*, 69–83. [CrossRef]

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).