Adaptive Computation Offloading with Task Scheduling Minimizing Reallocation in VANETs

Minyeong Gong¹, Younghwan Yoo² and Sanghyun Ahn³,*

¹ Department of Electrical and Computer Engineering, University of Seoul, Seoul 02504, Korea; gongmys@gmail.com
² School of Computer Engineering and Science, Pusan National University, Busan 46241, Korea; ymomo@pusan.ac.kr
³ Department of Computer Science and Engineering, University of Seoul, Seoul 02504, Korea
* Correspondence: ahn@uos.ac.kr

Abstract: Computation Offloading (CO) can accelerate application running through parallel processing. This paper proposes a method called *vehicular adaptive offloading* (VAO) in which vehicles in vehicular ad-hoc networks (VANETs) offload computationally intensive tasks to nearby vehicles by taking into account the dynamic topology change of VANETs. In CO, task scheduling has a huge impact on overall performance. After representing the precedence relationship between tasks with a directed acyclic graph (DAG), VAO in the CO requesting vehicle assigns tasks to neighbors so that it can minimize the probability of task reallocations caused by connection disruption between vehicles in VANETs. The simulation results showed that the proposed method decreases the number of reallocations by 45.4%, as compared with the well-known task scheduling algorithms HEFT and Max-Min. Accordingly, the schedule length of entire tasks belonging to one application is shortened by 14.4% on average.

Keywords: computation offloading; directed acyclic graph; task scheduling; vehicular ad hoc network

1. Introduction

As artificial intelligence (AI) has attracted much attention, many recent applications in vehicular ad-hoc networks (VANETs) also contain AI algorithms with resource-intensive computational tasks. However, all vehicles may not have enough computing resources to run those computationally intensive tasks, so computation offloading (CO) [1] has been investigated as a promising technique to overcome this resource limitation.

Historically, CO was proposed to lighten the computation overhead of resource-constrained devices by transferring computational tasks to resource-abundant servers like a cloud. However, since cloud servers are usually located in remote areas, long data transfer time hindered CO from being widely adopted in VANETs. The security concern along the long path from a requesting vehicle to a cloud server could also be another problem. This is why edge computing [2] and fog computing [3] were suggested, which perform computation at edge or fog servers near requesting vehicles.

However, edge or fog servers are in fixed locations, while vehicles move fast, so vehicles might leave the communication range of edge servers even before computation at the edge servers is complete. As a result, requesting vehicles should often receive computation results through multi-hop communication via road side units (RSUs), not directly from the edge servers. This multi-hop communication latency, although it is not as much as the latency from cloud servers, may be intolerable for recent time-sensitive applications, such as autonomous driving.

Therefore, this paper focuses only on offloading to nearby vehicles, instead of to fixed remote servers. CO to nearby vehicles has several advantages as compared with CO to edge servers. First, the distance to other vehicles is usually shorter than the distance to...
edge servers, which can reduce the bit error rate (BER) during the transmission. Second, the number of neighbor vehicles is usually greater than the number of neighbor edge servers, so more tasks can be offloaded at one time, leveraging the advantage of parallel processing.

However, there are two important issues to be solved in CO to other vehicles: (i) efficient task scheduling and adaptive allocation to service vehicles, (ii) reallocation of the tasks interrupted by topology changes.

In CO to cloud or edge servers, all computational works are statically assigned to several servers. On the other hand, there can be many candidate vehicles that can provide services in CO to other vehicles, so tasks should be efficiently scheduled and adaptively distributed to them. Actually, task scheduling has a huge impact on the overall performance of CO, but finding the optimal schedule is a well-known NP-hard problem \[4,5\]. Thus, heuristic scheduling methods should be designed.

When designing a task scheduling method, we must first consider the relationship between tasks. For example, if task B needs the results of task A, the two tasks should be assigned so that task A can be completed before task B. However, the precedence relationship between tasks may not be so explicit. Assuming that there is task C that needs the result of task B above, task C should also be executed after task A finishes. This kind of precedent and descendent relationship may not be seen at first glance.

We adopt a directed acyclic graph (DAG) to clearly show the precedence relationship between tasks. In a DAG, nodes represent task computation cost and edges represent the precedence relationship between tasks, as well as the data transfer cost. After constructing a DAG with tasks belonging to one application, a CO requesting vehicle (RV) assigns each task to a nearby vehicle, so that it may be completed before the required finish time. Then the actual finish time depends on the computing power of the CO service vehicles (SVs), so the information on the neighbors should be collected before the task allocation. Here, our proposed method, vehicular adaptive offloading (VAO), is used. VAO in an RV first collects the nearby vehicle information in real time, then allocates tasks to some of the neighbors so that the overall application can be completed as quickly as possible. Usually, the most computationally intensive task is assigned to the vehicle with the best computation power.

The allocation of tasks is not the end of the VAO operation. If any SV moves at a different speed, or in a different direction, from the RV, the SV may leave the communication range of the RV before completing its allocated task. In this case, the allocated task must be retrieved and reassigned to another vehicle. Then, the more task reallocation occurs, the longer the total execution time of the application usually is. So, the task allocation that can reduce the chance of reallocation is performed by VAO.

For performance evaluation, we conducted a series of simulations using the Veins simulator \[6\]. In terms of the schedule length (or the completion time of the application), VAO outperformed the existing task scheduling methods, HEFT \[7\] and Max-Min \[8\], by 14.4%. The task reallocation of VAO was also reduced by 45.4%.

The main contributions of this paper are as follows:

To the best of our knowledge, VAO is the first CO method to apply DAG scheduling to dynamic vehicular environments.

The proposed VAO can smoothly reallocate partial tasks for the completion of the whole workflow when an SV is out of the range of an RV. While HEFT and Max-Min are used only for static servers or processors, VAO ensures these scheduling algorithms deal with expected disconnection caused by velocity difference.

Before moving on to the next section, let us answer a question that many people ask about vehicle-to-vehicle (V2V) computation offloading: *Why does an SV provide the service to an RV?* Firstly, a single vehicle may not run the latest VANET applications alone. For example, object recognition-based driving assistance software can reduce blind spots and help in changing lanes, joining lanes and keeping distance safe \[9–11\], but it requires excessive computation from a single vehicle. If this kind of software can be run successfully through cooperation between vehicles and the object recognition result is shared between RVs and SVs, driving safety can be greatly improved. Secondly, SVs might get a direct
reward for providing services. For instance, blockchain or non-fungible token (NFT) technology can be used to create non-manipulable incentives to reward SVs. Since further discussion on blockchain and NFT is beyond the scope of this paper, we just assume that every vehicle in this paper is willing to cooperate with other vehicles.

The rest of the paper is organized as follows. In Section 2, related works on computation offloading and task scheduling are briefly overviewed. Section 3 presents the system model and Section 4 covers the numerical method to find the near-optimal task schedule in a static environment. The proposed VAO for a dynamic circumstance like VANET is described in detail in Section 5, and the simulation results are given in Section 6. Section 7 concludes this paper.

2. Related Works

This section overviews the offloading mechanism first, then explains famous task scheduling algorithms based on the DAG.

2.1. Computation Offloading

The authors of [1] proposed a deep Q-learning [12] based offloading mechanism for the mobile fog computing environment [1]. In their frameworks, an application is decomposed into basic blocks and offloaded to many devices. They evaluated latency and energy consumption for varying computation demands and a number of devices. MAUI [13], ThinkAir [14], and CloneCloud [15] are the frameworks for offloading to the mobile edge cloud. In the frameworks, the scheduling algorithm decides whether to offload tasks or not and deploys them according to the policy of the cloud. AVE [16] is an ant colony optimization (ACO) [17] based task offloading framework for mobile devices that are connected with each other by dedicated short-range communications (DSRC). It assumes that tasks are context-free, although the tasks of an application are actually tightly connected among themselves. Note that the precedence relationships and data transfer cost between tasks must be considered to optimize the completion time of an application. However, the works mentioned above model an application as one task or multiple context-free tasks.

CO to other devices requires data transfer overhead. The data transfer overhead is important, particularly when devices are connected with a wireless connection, like the vehicular environment. In the worst-case scenario, the data transfer time might overwhelm the task processing time, so several studies focused on reducing this overhead. For example, the authors of [18] proposed a method using federated learning (FL), in which local devices compute a deep learning (DL) model and a data center gathers the model updates. In [19], the authors suggested reducing the data transfer overhead of the FL method by introducing the random client selection mechanism. For that, federated distillation, based on knowledge distillation [20] and data augmentation [21], were adopted. However, the offloading schemes in [18–21] are very specific and only for deep-learning algorithms, so they cannot be applied to general applications.

OpenVDAP [22] classifies CO-possible services into four categories, (1) real-time diagnosis services, (2) driving assistance services, (3) in-vehicle infotainment services, and (4) third-party application services. It is very important, especially for real-time diagnosis and driving assistance services, to minimize the latency. The total latency of a service is related to the schedule length, but there has not been much work done on shortening the schedule length.

2.2. Task Scheduling

Task scheduling is one of the major factors affecting the performance of CO because the length of the schedule determines the total completion time of all tasks. Then, to the best of our knowledge, there have been no task scheduling methods that consider the precedence relationships between tasks in a dynamic topology like VANET. Thus, we
introduce two methods, HEFT [7] and Max-Min [8], instead, which are used in a static environment like parallel processing servers.

First, HEFT is a representative scheduling algorithm that considers the relationship between tasks in the form of a DAG, but it is basically for the multi-processor computing environment with very low communication cost between static processors. In HEFT, the earliest finish time (EFT) of each task, when it is executed on every processor, is calculated. For instance, regarding EFT \((t_i, p_j)\), the EFT of task \(t_i\) on processor \(p_j\) is computed as:

\[
EFT(t_i, p_j) = w_{ij} + \max_{m \in pred(t_i)} \left\{ \text{avail}[j], \max_{m \in pred(t_i)} (AFT(t_m) + c_{mi}) \right\}
\]

(1)

where \(w_{ij}\) is the expected execution time of the task \(t_i\) on processor \(p_j\); \(\text{avail}[j]\) is the earliest time when \(p_j\) is ready to execute \(t_i\); \(\text{pred}(t_i)\) is the set of predecessor tasks of \(t_i\); \(AFT(t_m)\) is the actual finish time (AFT) of \(t_m\); and \(c_{mi}\) is the data transfer cost from \(t_m\) to \(t_i\). The schedule length is defined as the largest one among AFTs of all tasks:

\[
\text{schedule length} = \max_{t_i} \{ AFT(t_i) \}
\]

(2)

HEFT schedules a set of tasks, \(T\), in a DAG \(D\) on a given set of heterogeneous processors \(P\). HEFT computes the average processing cost \(\bar{\pi}_i\) of a task \(t_i\) and the average communication cost \(\tau_{ij}\) between tasks \(t_i\) and \(t_j\) in the given DAG. Then, it calculates the upward ranks of every task in \(D\) reversely from an exit task \(t_{exit}\). The upward rank \(rank_u(t_i)\) is the total cost from task \(t_i\) to the exit task, defined as:

\[
rank_u(t_i) = \bar{\pi}_i + \max_{t_h \in isu(t_i)} \left( \tau_{ih} + rank_u(t_h) \right)
\]

(3)

where \(isu(t_i)\) is the set of the immediate successors of \(t_i\). HEFT sorts the tasks in \(T\) in non-increasing order by \(rank_u\) into the sorted list \(T'\). Then, it gets the first task \(t_i\) (i.e., the task with the highest rank) \(t_i\) in \(T'\) and allocates the task to the minimum EFT processor (i.e., the processor that can finish the task the fastest) using insertion-based scheduling. The insertion-based scheduling inserts \(t_i\) to the first time slot of \(p_k\) where the task can be executed. Finally, \(t_i\) is allocated to the minimum EFT processor \(p_{min}\).

Another well-known scheduling method, Max-Min [8], is very simple, considering only the computation power of processors. Max-Min calculates the estimated finish time \(K_{ij}\) of a task \(t_i\) on a processor \(p_j\) for every task and every processor. Then, Max-Min allocates the task with the largest \(K_{ij}\) to the processor that can execute it the fastest. This allocation is repeated until every task is scheduled.

Several recent works [23,24] utilize duplication-based DAG scheduling. The Task Duplication-based Clustering Algorithm (TDCA) [23] improved performance by accurate parameter definition, enhanced clustering, and by considering the chain reaction. On the other hand, Lachesis [24], which is a task duplication-based learning algorithm, uses a convolutional network to parse a DAG and uses a heuristic search algorithm to allocate selected tasks to selected nodes. Although duplication-based scheduling is effective in reducing the schedule length, by maximizing computing resource usage, it cannot work on resource-constraint devices, such as mobile phones and vehicles, since it has to make copies of tasks to idle processors.

Table 1. Comparison of Related Works.

| Solutions | Task Granularity | Network Topology | Computation Provider |
|-----------|-----------------|-----------------|---------------------|
| AVE [16]  | Individual      | static          | edge                |
| OpenVDAP [22] | Individual   | static          | edge/vehicular      |
| HEFT [7]   | DAG             | static          | processor           |
| Max-Min [8] | Individual     | static          | processor           |
| TDCA [23]  | DAG             | static          | processor           |
| Lachesis [24] | DAG           | static          | processor           |
| VAO [this work] | DAG       | dynamic         | vehicular           |
3. System Model

Offloading tasks to clouds or other devices allows mobile devices to overcome the limitation on computation resources. However, offloading to the remote cloud is not suitable for real-time applications because of inevitable long latency. Even in the case of edge/fog computing, a fast-moving vehicle can easily go far from the edge/fog server in a short time, resulting in increased latency. According to current vehicular technology trends, we may expect that vehicles are to be equipped with computational resources, such as CPU, memory, and communication capabilities, and will be able to execute applications using their computational resources managed by virtual machine techniques [25,26]. In this environment, a vehicle can request other vehicles to handle its tasks. In modern computer systems, various types of high-performance computing processors, like CPU, graphics processing unit (GPU), neural processing unit (NPU), field programmable gate array (FPGA) utilizing CUDA [27] or OpenCL [28] can be used to process codes. In this paper, regardless of the types of processors, we model the computing power of a vehicle in MIPS (million instructions per second) units.

Vehicles in a VANET communicate with each other via V2V communications, such as DSRC and cellular vehicle-to-everything (C-V2X). After searching for nearby vehicles, an RV requests SVs to execute its tasks. A set of SVs, $V'$, is a subset of the set of searched vehicles $V$ ($V' \subseteq V$). Assuming that there is no conflict and fading on wireless channels between vehicles, we consider only the amount of data that should be transferred on each channel. Then, the communication overhead on wireless is usually larger than the wired channel, so the transfer cost should be complemented by reducing the processing cost with the optimized task schedule.

Figure 1 compares the efficiency of the task schedules made by HEFT and Max-Min. Suppose that an application consists of five tasks and their precedence relationships can be represented in a DAG as in Figure 1a. In this graph, nodes and edges correspond to tasks and the dependency between two tasks, respectively. If a task $t_i$ in a DAG needs the result of another task $t_k$, $t_k$ must be finished before the execution of $t_i$. In this case, $t_k$ is the predecessor of $t_i$ and $t_i$ is the successor of $t_k$. An edge of a DAG points from a predecessor to a successor.

Figure 1b,c are the Gantt charts of task executions by Max-Min and HEFT. Here, $v_1$ is an RV and the other two vehicles are SVs; and the computation powers of $v_1$, $v_2$, and $v_3$ are 100, 150, and 200 MIPS, respectively. Max-Min and HEFT in $v_1$ allocate tasks according to their algorithms. As you can see, HEFT made a shorter schedule length than Max-Min by overlapping data transfer time with execution time on SVs. In our experiment, the schedule length of HEFT was shorter than that of Max-Min by 15.3% on average. This is because HEFT computes the explicit rank of each task, in other words, the order of executing each task, in an upward direction from an exit task based on a DAG, while Max-Min does not use a DAG. Therefore, we also adopt the DAG for the proposed method.

![Figure 1](image_url)
In this example, the task allocation is decided offline before actual execution, since HEFT and Max-Min were designed for static environments, such as parallel processing servers. However, in VANET, task reallocation must be handled frequently, since the associated SVs may leave the communication range of the RV without returning the computation result.

4. Problem Formulation

As mentioned earlier, the optimal task allocation problem is NP-hard. However, we can get a near-optimal solution using linear programming.

First, the allocation of \( t_i \) is denoted by a binary variable \( x_{ij} \) which is set to 1 when \( t_i \) is allocated to vehicle \( v_j \), and otherwise, 0:

\[
x_{ij} = \begin{cases} 
1 & \text{if } t_i \text{ is allocated to } v_j \\
0 & \text{otherwise}
\end{cases}
\] (4)

A task schedule \( \chi \) is represented by a matrix consisting of \( x_{ij} \)'s:

\[
\chi = \begin{bmatrix} 
    x_{11} & \cdots & x_{1n_v} \\
    \vdots & \ddots & \vdots \\
    x_{n_t1} & \cdots & x_{n_t n_v}
\end{bmatrix} 
\] (5)

where \( n_v \) is the number of SVs and \( n_t \) is the number of tasks.

For the sake of simplicity, we assume that a vehicle executes only one task at a time and that a task cannot be allocated to more than one vehicle simultaneously. In order to guarantee a task being allocated to only one vehicle, the following condition must be satisfied:

\[
x_{11} + x_{12} + \ldots + x_{1n_v} = 1, \forall i \in \{1, 2, \ldots, n_t\} \] (6)

A task without any predecessor is called an entry task, and a task without any successor is called an exit task. Only the applications with one entry task are considered in this paper.

For executing task \( t_i \) on an SV \( v_j \), all of its predecessors must be finished beforehand, and the results from the predecessors must be received by \( v_j \). Therefore, \( ST(t_i) \), the start time of \( t_i \), must satisfy the following condition:

\[
ST(t_i) > \max_{t_m \in \text{pred}(t_i)} (AFT(t_m) + c_{mi}) 
\] (7)

where \( \text{pred}(t_i) \) is the set of the immediate predecessors of \( t_i \) and \( c_{mi} \) is the data transfer cost from \( t_m \) to \( t_i \). Furthermore, because an SV is assumed not to execute more than one task at the same time, \( ST(t_i) \) must satisfy the following condition:

\[
ST(t_i) < ST(t_k) \text{ or } ST(t_i) > AFT(t_k), \ t_k \in \text{alloc}(j) 
\] (8)
where \( AFT(t_k) = ST(t_k) + E_{kj} \) and \( alloc(j) \) is the set of tasks allocated (or offloaded) to \( v_j \). The value \( E_{kj} \) is the execution time of \( t_k \) on \( v_j \) and can be computed as \( E_{kj} = O(k) \frac{ipb(k)}{\text{mips}(j)} \) where \( O(k) \) is the data size of \( t_k \), \( ipb(k) \) is the number of instructions needed to process 1 byte of data in \( t_k \), and \( \text{mips}(j) \) is the computing power of processor \( j \) expressed in MIPS.

Assuming that there is no task reallocation, the objective function is:

\[
\min_{t_i \in T} \max_{t_i \in RT} \{ AFT(t_i) \}
\]

\[ \text{s.t. (4)} \sim (8) \]

However, a fast-moving SV can be out of the communication range of the RV easily, so unfinished tasks that had been allocated to the SV have to be reallocated to other vehicles. If the set of the tasks that should be reallocated is \( RT \), the objective function is:

\[
\min_{t_i \in RT} \max_{t_i \in RT} \{ AFT(t_i) \}
\]

\[ \text{s.t. (4)} \sim (8) \]

In the dynamic vehicular environment, the vehicles in the coverage area of a vehicle tend to change frequently. After offloading tasks to SVs, the RV has to continuously track the SVs for possible task reallocation events.

5. Vehicular Adaptive Offloading (VAO)

While linear optimization on the task schedule is possible in a static environment, a dynamic situation like a VANET requires a heuristic method, since even the information on the service processors is not basically given. This section firstly overviews the operation of our method of VAO, then explains the proposed task scheduling and task reallocation scheme in detail.

5.1. Operation Overview

Figure 2 shows the operation of VAO. First, an RV \( v_r \) searches for nearby vehicles and collects their information, such as computing power and route, by broadcasting a message. Then \( v_r \) assigns computational tasks to nearby vehicles according to the task schedule it makes, based on neighbor vehicle information. After that, it periodically checks whether SVs will be out of its communication range soon, and in that case, unfinished tasks on the SVs are reallocated to other vehicles. This procedure is depicted in Algorithm 1. Before delving into the algorithm, see Table 2 for the notations.

![Figure 2](image-url)
Table 2. The notations and meanings.

| Notation | Meaning |
|----------|---------|
| $D$      | a directed acyclic graph |
| $V$      | a set of searched vehicles |
| $V^s$    | a set of service vehicles |
| $v_r$    | a requesting vehicle (RV) |
| $v_s$    | a service vehicle (SV) |
| $v_{\text{min}}$ | an SV that can provide the minimum EFT for a task |
| $T$      | a set of the tasks |
| $t_i$    | a task |
| $t_{\text{exit}}$ | an exit task |
| $\chi$  | a matrix of a task schedule |

Algorithm 1: VAO in a RV $v_r$

Input: a given DAG $D$, a set of tasks $T$ in $D$

1. initialize a set of target tasks $T'$ to be the same with $T$
2. let $V$ be a set of searched vehicles
3. let $V^s$ be a set of SVs in $V$ ($V^s \subseteq V$)
4. set $V \leftarrow \emptyset$
5. repeat
6. search vehicles
7. add every searched vehicle to $V$
8. $\chi, V^s \leftarrow \text{task_schedule}(T', V)$
9. for every task $t_i$ in $T'$
10. transfer $t_i$ to the vehicles in $V^s$ according to $\chi$
11. remove $t_i$ from $T'$
12. end for
13. start/resume execution of the application by transferring the input data
14. $T' \leftarrow \text{check_realloc}(V^s)$
15. until all SVs complete the allocated tasks

Algorithm 1 describes the operation of VAO in an RV $v_r$. After initialization, $v_r$ conducts a search broadcasting a vehicle information request (VIRQ) message, and each vehicle receiving this message sends a vehicle information reply (VIRP) with the information on its route and computing resources (Line 6). The searched vehicle information is stored into $V$ (Line 7), then $v_r$ calls up the task scheduling algorithm to determine $\chi$ based on the information on the set of the tasks $T$ and the computing power of vehicles in $V$ (Line 8). This task scheduling algorithm is described in Algorithm 2 in the next section. Then, $v_r$ transfers tasks in $T$ to the vehicles in $V^s$ according to $\chi$ (Line 9 to 12). Then, in Line 13, $v_r$ starts to execute the application by transferring input data to the SVs that have entry tasks.

After completing its allocated tasks, an SV transmits the result to the SVs that have successor tasks. However, in a VANET where vehicles move fast, SVs may frequently leave the communication range of the RV before completing their work, and this can cause a big problem in the CO system because all successor tasks cannot be executed until they receive the results of predecessor tasks. Therefore, we need to reallocate the tasks associated with original SVs that have gone far away without completing the task. If any SV not finishing its assigned tasks has gone too far apart from $v_r$, those tasks are reallocated to other nearby vehicles. This check and reallocation are periodically done until all offloaded tasks are complete. The detailed procedure will be explained in Section 5.3.

5.2. Task Scheduling

The proposed task scheduling method has two important features: first, it utilizes the DAG to consider the precedence relationships between tasks, and, second, it selects only nearby vehicles whose routes are similar to the route of RV as SVs, in order to reduce the
chance of task reallocation. As mentioned earlier, a task schedule $\chi$ is the most important factor affecting performance, since a long task schedule means a long execution time. In general, the more reallocations that occur, the longer the task schedule is.

Algorithm 2: task schedule $(T, V)$

```
Input: a given DAG $D$, a set of tasks $T$ in $D$, a set of searched vehicles $V$
Output: task schedule $\chi$
1 compute average processing and communication cost
2 calculate rank$_u$ of every task in $D$ in an upward direction, starting from an exit task
3 let $T'$ be the list of tasks in $T$, sorted in non-increasing order by rank$_u$
4 $V' \leftarrow \{v_i | v_i \in V$ and its route overlaps the route of $v_r$ by more than $\alpha$ blocks$\}$
5 while there exist tasks in $T'$ do
6 $t_i \leftarrow$ the first task in $T'$
7 for $v_k$ in $V'$ do
8 calculate the EFT $(t_i, v_k)$ with insertion-based scheduling
9 end for
10 allocate $t_i$ to the minimum EFT vehicle $v_{min}$
11 remove $t_i$ from $T'$
12 end while
```

Lines 1 to 2 compute the average cost of searched vehicles and the rank$_u$ of every task in an upward direction from an exit task. Line 3 sorts the tasks in non-increasing order by rank$_u$. Then, in Line 4, out of the searched vehicles, only vehicles whose routes overlap the route of $v_r$ by more than $\alpha$ blocks are filtered into a new set of vehicles $V'$. Only vehicles in $V'$ are considered as a candidate SV that can provide the minimum EFT for each task in subsequent lines. Line 6 gets the first task in the sorted list $T'$ into $t_i$. In Lines 7 to 9, $v_r$ calculates the EFT of $t_i$ on every $v_k$ in $V'$ with insertion-based scheduling. Line 10 allocates $t_i$ to the vehicle that can provide the minimum EFT. Lines 6 to 11 are repeated while there exist any unallocated tasks.

5.3. Check Reallocation

Although our scheduling method tries its best to avoid task reallocation, this cannot be perfectly avoided because of the dynamic topology change in VANETs. The following Algorithm 3 shows how VAO handles task reallocation caused by the connection disruption topology change makes.

Algorithm 3: check realloc $(V')$

```
Input: a set of SVs $V'$
Output: a set of unfinished tasks $T$
1 periodically broadcasts RV's location information
2 if receives a message that $v_s$ are expected to leave the communication range
3 informs all SVs of reallocation event to make them stop data transfer to $v_s$
4 let $T$ a set of unfinished tasks allocated to $v_s$
5 request $v_s$ to transfer the stored data for the tasks
6 end if
7 return $T$
```

In Line 1, an RV $v_r$ periodically broadcasts its location information to SVs, then SVs which are expected to be out of communication range send report messages to $v_r$. In Lines 2 to 6, all SVs are informed of a task reallocation event if any $v_s$ is expected to be out of range. All the SVs stop data transfer to $v_s$ and $v_r$ requests $v_s$ to transfer stored data for the tasks. After that, the algorithm returns the set of unfinished tasks, so Algorithm 1 can reallocate the tasks.

To filter out the vehicles, Algorithm 2 needs to compare every route of the searched vehicles with that of RV. The time complexity of comparing the routes is $O(|Y|)$ when the
number of the searched vehicles is $|V|$. The time complexity of finding the first idle time slot is $O (|T|^2 |V|)$. So, the total complexity of the VAO is $O (|T|^2 |V|)$.

6. Performance Evaluation

Using the Veins network simulator [6], we compared the performances of VAO, HEFT, and Max-Min in terms of schedule length, reallocation probability, and data transfer time. Actually, since neither HEFT nor Max-Min include a way to reallocate tasks, the VIRQ/VIRP probability of each algorithm is investigated. Finally, the effect of the depth of width of a process of VAO was adopted for the HEFT and Max-Min task reallocation. So, the message mentioned the application of VAO, HEFT, and Max-Min in terms of schedule length, reallocation probability, and data transfer time. On the other hand, VAO created schedules 11.7% longer than HEFT, although the schedule considering relationships between tasks of an application, which is represented in a DAG. On the other hand, VAO created schedules 11.7% longer than HEFT, although VAO also utilizes a DAG. This is because VAO filters out some searched vehicles from the set of candidate SVs, based on how long their routes overlap the route of the RV. The smaller the number of candidate SVs, the smaller the number of tasks that can be executed in parallel. Furthermore, if some vehicles with high computing power are excluded from the candidate SVs’ set, the total computing capability of the offloading system would be degraded. More experiments were needed to better understand these results.

The road structure is shown in Figure 4. A total of 100–300 vehicles, which is proportional to the simulation time, with a maximum speed of 30 km/h were randomly deployed on the roads. Each of them followed one of 30 routes that were randomly generated, and its processing power followed uniform distribution between 200 and 600 MIPS.

Figure 5 compares the average schedule length over 1000 simulations for an application that can be represented by a DAG with both depth and width of 4. As mentioned in Section 3, the schedule length by HEFT was 15.3% shorter than Max-Min, since HEFT makes the schedule considering relationships between tasks of an application, which is represented in a DAG. On the other hand, VAO created schedules 11.7% longer than HEFT, although VAO also utilizes a DAG. This is because VAO filters out some searched vehicles from the set of candidate SVs, based on how long their routes overlap the route of the RV. The smaller the number of candidate SVs, the smaller the number of tasks that can be executed in parallel. Furthermore, if some vehicles with high computing power are excluded from the candidate SVs’ set, the total computing capability of the offloading system would be degraded. More experiments were needed to better understand these results.
which is twice the width of the DAG. This means if there are sufficient vehicles, VAO can improve performance because some of the searched vehicles were excluded from the candidate hand, generated, and mentioned proportional to their number. Figure 4 shows the road environment of the Veins simulation. The black line is the road and red triangles are vehicles.

![Road environment of the Veins simulation](image)

**Figure 4.** The road environment of the Veins simulation. The black line is road and red triangles are vehicles.

![Schedule length comparison](image)

**Figure 5.** Initial schedule length without reallocation for a depth 4 and width 4 DAG application.

We performed additional simulations 1000 times with various densities of vehicles so that the number of searched vehicles could be different. Figure 6a depicts the schedule length against the number of searched vehicles $|V|$. When $|V|$ was 0, tasks could not be offloaded (no CO happened); and when $|V|$ became 1, the schedule lengths of HEFT and Max-Min decreased by 39.4%. As $|V|$ increased, the schedule length typically decreased, but after $|V|$ was greater than 4, the schedule lengths by HEFT and Max-Min were stable around 41.7% and 49.1% of the no CO case. This is because the width of the DAG for our application was 4. Since up to four tasks can be executed simultaneously in a level, many more vehicles than 4 are not much help to improve parallel computing efficiency. However, in the case of VAO, about twice as many searched vehicles were needed to stabilize the performance because some of the searched vehicles were excluded from the candidate SVs’ set, due to their different routes from the RV’s route. Figure 6b shows the schedule length when the number of searched vehicles $|V|$ is sufficient. When $|V|$ is large enough, the schedule lengths by HEFT and Max-Min do not change much. On the other hand, the schedule length by VAO continuously decreases as $|V|$ increases.

Figure 6c shows the number of SVs according to the number of searched vehicles. Actually, we can see that the number of vehicles that provide computation services in HEFT became equal to the number of SVs in HEFT when the number of searched vehicles was 8, which is twice the width of the DAG. This means if there are sufficient vehicles, VAO can give good performance.
The accumulated data transfer times during the execution of an application are presented in Figure 6d. While the data transfer time of Max-Min continuously increased, the data transfer times of HEFT and VAO barely increased once stable. This is because Max-Min considers only computation power of vehicles in selecting SVs, resulting in task offloading to an unnecessarily excessive number of SVs as \(|V|\) increases. On the contrary, HEFT and VAO select SVs considering data transfer time as well as computation power of vehicles in (1) and (3). Thus, although a neighbor vehicle has a little higher computation power, an RV does not offload tasks to the neighbor if the data transfer cost is too high.

As seen in Figure 5, the initial schedule length by VAO is slightly longer than HEFT if there are not enough vehicles around an RV because some neighbor vehicles are excluded from the candidate SVs’ set. However, in terms of reallocation overhead, VAO is better than others.

Figure 7 compares the task reallocation probability of the three methods. This result comes from the same simulations in Figure 6. Only 15.2% of tasks were reallocated to others in VAO, whereas 60.6% and 86.2% of tasks had to be reallocated to others in HEFT and Max-Min, respectively. As a result, the schedule length by VAO, including reallocated tasks, was shorter than HEFT by 14.4%, although the initial schedule length by VAO was 11.7% longer than HEFT.

The performance of the CO scheme can be different according to the depth and width of a DAG. Figure 8a,b illustrate initial schedule lengths associated with varying the depth and width of a DAG. While one of the width and depth was varied, the other was fixed at 4. In both graphs, as the depth or the width increases, the length of the schedules increases naturally, since the total number of tasks increases.

Comparing “large depth” to “large width”, the large width was better to create shorter schedule lengths because more tasks can be executed in parallel on more SVs. Figure 8b shows that Max-Min is also a good option in the case where the width is large. However, the width of a DAG can be widened only when tasks have no tight relationships with each other. If tasks have strict precedence relationships between themselves, only the depth can increase. Figure 8a shows that the schedule length difference between Max-Min and the other methods got bigger as the depth of a DAG increased. When the depth was 8, the schedule length difference between HEFT and Max-Min was 14.8 s, 7.7 times larger than the difference when the depth was 2. This substantiates the view that DAG scheduling is efficient for applications with a large depth.

Figure 6. Cont.
Figure 6. Schedule length and data transfer time for a depth 4 and width 4 DAG application: (a) Schedule length against the number of searched vehicles; (b) Schedule length against the number of searched vehicles; (c) The number of SVs against the number of searched vehicles; (d) Data transfer time against the number of searched vehicles.
Figure 7. Probability of task reallocation.

Figure 8. Initial schedule length according to the shape of the DAG: (a) Schedule length with varying depth of DAG; (b) Schedule length with varying width of DAG.

Figure 9 represents the data transfer time with varying depth and width. Figure 9a shows the data transfer time of Max-Min was longer than that of HEFT and VAO by 18–20%. Meanwhile, Figure 9b shows that the data transfer times of HEFT and Max-Min became closer as width increased, and the difference was just 5.8% when the width was 8. Anyway, in both graphs, VAO outperformed the two other methods, since it selects a relatively smaller number of SVs, taking both their routes and data transfer time into account.

Figure 9. Data transfer time with varying depth and width of DAG: (a) Data transfer time against the depth of DAG; (b) Data transfer time against the width of DAG.
7. Conclusions

We proposed vehicular adaptive offloading (VAO) for the dynamic vehicular environment, which uses DAG scheduling for optimizing task execution. Unlike other well-known scheduling methods, HEFT and Max-Min, VAO focuses on reducing task reallocation, taking into account the dynamic topology of VANETs. For that purpose, VAO does not offload tasks to neighbors whose routes are much different from the RV’s route. The simulation results showed that VAO decreases the probability of task reallocation by 45.4%, as compared with HEFT. Our simulation reveals that more SVs are required than the width of DAG to optimize CO but many more vehicles than the width are not helpful. When the number of searched vehicles is sufficient, the schedule length of VAO, including all reallocated tasks, is 14.4% shorter than that of HEFT.

We have two future works. First, during the simulation, we occasionally observed that the schedule length of a task set increased when CO was used, as compared with the case where all the tasks were performed in one vehicle. This is mainly because of unexpected task reallocation events, so, if we can predict the chance of task reallocation occurring, inefficient CO can be avoided. We think that AI methods like LSTM (Long Short-Term Memory), which can analyze time-series data on number and distribution of vehicles, might be helpful to predict task reallocation events. Second, we did not consider the situation where multiple vehicles request CO at the same time, resulting in possible conflict at their shared nearby vehicles. This might be solved if a nearby vehicle informs each CO RV of the computing power that can be provided to the RV exclusively.

Table 3 summarizes the characteristics of the algorithms observed in this paper.

Table 3. Characteristics of task scheduling algorithms.

| Algorithm | Characteristics |
|-----------|----------------|
| HEFT      | DAG scheduling  |
|           | Good performance in a static environment |
| Max-Min   | Considering only computation power |
|           | High data transfer overhead |
|           | Good performance if tasks in an application are independent |
| VAO       | DAG scheduling  |
|           | Lower reallocation probability |
|           | Good performance in a dynamic environment |

Author Contributions: Conceptualization, M.G. and S.A.; methodology, M.G.; software, M.G.; validation, M.G., Y.Y. and S.A.; formal analysis, Y.Y.; investigation, M.G.; resources, M.G.; data curation, M.G.; writing—original draft preparation, M.G.; writing—review and editing, Y.Y.; visualization, M.G.; supervision, S.A.; project administration, S.A.; funding acquisition, S.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1A2C1011184).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References
1. Alam, M.G.R.; Hassan, M.M.; Uddin, M.Z.; Almogren, A.; Fortino, G. Autonomic Computation Offloading in Mobile Edge for IoT Applications. *Future Gener. Comput. Syst.* 2019, 90, 149–157. [CrossRef]
2. Patel, M.; Naughton, B.; Chan, C.; Sprecher, N.; Abeta, S.; Neal, A. Mobile-Edge Computing Introductory Technical White Paper. *White Pap. Mob. Edge Comput. (MEC) Ind. Initiat.* 2014, 29, 854–864.
3. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and Its Role in the Internet of Things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012.
