Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 2033644, 14 pages
https://doi.org/10.1155/2022/2033644

Research Article

Multihop Transmission-Oriented Dynamic Workflow Scheduling in Vehicular Cloud

Qiang Zhang

College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China

Correspondence should be addressed to Qiang Zhang; cszhangqiang@nuaa.edu.cn

Received 11 August 2022; Revised 22 November 2022; Accepted 26 November 2022; Published 8 December 2022

Academic Editor: Mohd Dilshad Ansari

Vehicular cloud as a kind of resource aggregation can provide cloud service flexibly for intelligent transportation systems and individual users. However, the dynamicity of vehicular cloud such as dynamic multihop transmission also brings new challenges to workflow scheduling. In this paper, a heuristic dynamic workflow scheduling (HDWS) strategy is proposed to solve the problem of workflow scheduling with dynamic multihop transmission in vehicular cloud. HDWS establishes a local scheduling based on the current resource status and task sets. The makespan of local scheduling is optimized by reassigning tasks based on task interdependency. At run time of an application, HDWS estimates the makespan in real time based on the current transmission rates and the unready tasks will be rescheduled once it detects the degradation of the makespan. Experimental results show that the proposed HDWS can improve the service success ratio of vehicular cloud and decrease the makespan compared to the existing approaches.

1. Introduction

With the advancement of vehicular ad hoc networks (VANETs) and cloud computing, smart vehicles with certain resources constitute vehicular cloud (VC) [1]. VC as an emerging service paradigm has become an important part of intelligent transportation systems (ITS) and also increases the available resources for the fixed cloud such as central cloud and edge cloudlet. The basic definition of VC can be described as "*A group of vehicles whose corporate computing, sensing, communication and physical resources can be coordinated and dynamically allocated to authorized users*" [2]. A vehicle in VC can be viewed as a vehicular node that typically owns an on-board computer, a GPS device, a radio transceiver, several radar devices, several cameras, some sensors, and so on. VC can integrate and schedule resources to perform applications or achieve some functions such as collision avoidance, intersection signal control, and automated traffic routing for intelligent transportation systems [3–5]. Besides, VC also can be utilized as a cooperator of central cloud and edge cloudlet. For instance, the edge cloudlet connected with the base station or roadside unit can offload applications or tasks to VC or vice versa [6, 7]. The mobility of vehicles in VC makes its resource provisioning volatile, and the volatility should be considered for resource scheduling in VC [8]. For example, vehicles in VC may depart from the current vehicular ad hoc network due to moving speed difference, and then, the corresponding resources also become unavailable. Conversely, when vehicles enter the VC, the amount of available resources is increased. In addition, the dynamic network topology of VC leads to the volatility of the network resource. For instance, the vehicle-to-vehicle communication is achieved through one hop or multihop transmission in VC and the corresponding transmission rate is dynamic owing to the change of the shortest transmission path. The implications are two-fold. On the one hand, mobile and distributed resources of vehicles can be integrated to create various resources combinational provisioning to expand cloud services. On the other hand, the dynamicity of resources makes difficulties for task scheduling in VC. How to efficiently utilize resources and meanwhile handle the dynamicity of resources well in VC remains to be an open challenge.

VC is a heterogeneous computing system that contains diverse resources interconnected by the dynamic vehicular ad hoc network. Some applications supported by VC are classified into the workflow application considering the interdependency among tasks. For example, 3D object detection and augmented reality for autonomous driving are workflow applications [9]. Specifically, a 3D object detection application usually consists of a 2D detector, depth generator, data transformation, segmentation, and det-net and the entire application can be regarded as a workflow application that contains dependent tasks [10]. A workflow application can be modeled by a directed acyclic graph in which the vertexes represent tasks and edges represent dependencies among tasks. The general workflow scheduling needs to assign tasks to suitable vehicular nodes and order task executions on each vehicular node. Meanwhile, task dependencies are satisfied and one or more performance metrics such as the makespan and cost are optimized. The fundamental workflow scheduling problem is NP-complete [11]. Compared to independent task scheduling, data transmissions among dependent tasks are required by workflow scheduling when they are assigned to different vehicular nodes. In conventional cloud computing such as a datacenter, the transmission rates between servers are usually stable. However, the dynamicity of the multihop transmission path in vehicular networks leads to the dynamic transmission rate so that the transmission time is fluctuating. Since it is difficult to estimate all transmission time in priori with low time complexity in VC, designing an efficient workflow scheduling considering dynamic mutlihop transmission is nontrivial.

In this paper, a heuristic dynamic workflow scheduling (HDWS) strategy is proposed to solve the problem of workflow scheduling in VC under the condition of dynamic multihop transmission. HDWS classifies tasks into task sets based on task dependencies and partitions the given deadline into several time segments. Several worker node sets are created according to time segments and task completion time. Subsequently, a local scheduling is established based on task sets and worker node sets. Two scheduling algorithms are proposed to optimize local scheduling in the aspect of the makespan. HDWS detects current transmission rates in real time so as to estimate the makespan. If the estimated makespan is longer than the deadline, the unready tasks will be rescheduled by HDWS according to the current situation.

The main contributions of this paper are listed as follows:

(1) To our best knowledge, this is the first work to study the workflow scheduling considering dynamic multihop transmission in VC

(2) A multihop transmission-oriented dynamic workflow scheduling strategy in VC is designed to cope with the dynamicity of transmission rates and further optimize the makespan

(3) HDWS contains static scheduling and dynamic scheduling. The static scheduling creates local scheduling based on divide and conquer and then improves the local scheduling based on the interdependency among tasks. The dynamic scheduling detects transmission rates in real time and reschedules the unready tasks based on the current system state. Besides, HDWS achieves better performance compared to the existing method [6]

The remainder of this paper is organized as follows. In the next section, the related work is summarized. Section 3 presents the system model and problem formulation. The multihop transmission-oriented dynamic workflow scheduling strategy is proposed in Section 4. Section 5 elaborates performance evaluation. Finally, the conclusion is given in Section 6.

2. Related Work

According to the service provider, the researches on workflow scheduling in cloud computing can be classified into the following scenarios.

2.1. Workflow Scheduling in Central Cloud. The typical scenario in this category mainly involves a workflow application and multiple virtual machines, and workflow tasks are allocated to virtual machines to speed up execution with acceptable or a given cost. Calheiros and Buyya proposed an algorithm that uses idle time of provisioned resources and budget surplus to replicate tasks so as to increase the probability of meeting application deadlines [12]. Zhu et al. proposed an evolutionary multiobjective optimization-based algorithm with a novel encoding scheme to solve the multiobjective workflow scheduling problem which minimizes both the makespan and the cost simultaneously [13]. Kaur and Mehta presented an augmented shuffled frog leaping algorithm-based solution for workflow scheduling in the cloud environment to optimize the cost while meeting the specified deadline [14]. Wu et al. proposed a heuristic task deployment and scheduling algorithm that finds the minimum number of virtual machine instances needed to guarantee an application's deadline and also minimizes the makespan of the application [15]. Sahni and Vidyarthi proposed a dynamic cost-effective deadline-constrained heuristic algorithm for scheduling a scientific workflow in a public cloud considering virtual machine performance variability [16]. However, in this scenario, transmission rates among virtual machines are usually stable during application running. The above methods do not consider dynamic transmission rates and are not suitable for workflow scheduling in VC.

2.2. Workflow Scheduling in VC. Sun et al. proposed a modified genetic algorithm-based solution to achieve workflow scheduling in VC, and the proposed encoding scheme utilizes the fixed time slot [6]. Qi et al. proposed a deep reinforcement learning-based resource scheduling policy in vehicular edge computing [17]. Liu et al. proposed an efficient task scheduling algorithm to prioritize multiple applications and prioritize multiple tasks so as to guarantee the completion time constraints of applications [18]. Ku et al. proposed an application-adaptive task partitioning and offloading algorithm to minimize the end-to-end delay and maximize application level performance [19]. But the aforementioned solutions ignore the dynamicity

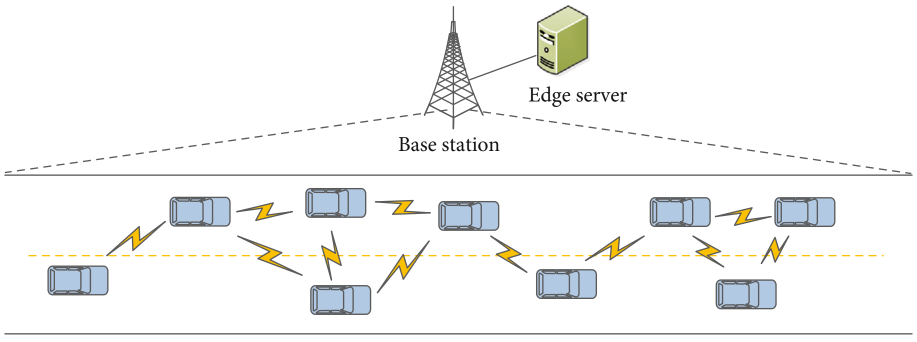


Figure 1: System architecture of VC.

of multihop transmission among vehicles in VC, so they are not applicable to the scenario of dynamic multihop transmission.

Besides the above scenarios, the researches in which a mobile user using a smartphone or a vehicle offloads part of its workflow tasks to one or more edge-fixed cloudlets also involve workflow scheduling [20–22]. In this scenario, the resource nodes usually include fixed edge servers and a user device and the workflow application can be cooperatively executed by them. This scenario does not involve the dynamic multihop transmission and cooperation among mobile vehicles, and consequently, these schemes are also not appropriate for the scenario of dynamic multihop transmission. In addition, some researchers investigate uncertainty of tasks in workflow scheduling [23, 24]. For example, the task execution time and the size of output data cannot be accurately predicted before scheduling; the arrival of new tasks results in fluctuation in task execution time.

The above researches focus on workflow scheduling in a stable and certain environment or the uncertainty of task in scheduling; the dynamicity of network resource is not well covered. How to tackle dynamic transmission time caused by node mobility and implement an efficient workflow scheduling to minimize the makespan while satisfying the stability of the service in a dynamic vehicular cloud system is the new challenge.

3. System Model and Problem Formulation

3.1. System Architecture. As shown in Figure 1, this paper mainly considers a VC scenario in which vehicular nodes within the coverage of a base station (BS) provide cloud services. An edge server is typically connected with BS. The edge server and the corresponding BS constitute a roadside unit (RSU). Hence, vehicular nodes can exchange information with RSU through the on-board unit (OBU) and wireless network [25]. When a vehicle enters into the communication range of the BS, it will transmit its position, velocity, and computation capability to the BS. The VC in a specific area is maintained by the corresponding BS. Nodes in VC can communicate with each other by the vehicular multihop network as well as the BS. The input data of an application is firstly sent by the BS to vehicular nodes, and then, the workflow tasks are executed by vehicular nodes. In order to provide high-quality cloud services, the edge server needs to collect status information about VC in real time to achieve an efficient task scheduling.

3.2. Mobility and Communication Model. This paper mainly discusses the expressway environment in which vehicles move almost at constant speeds within a relative long distance. Vehicles at different positions have different download rates from the BS. A vehicle has a higher download rate when it is closer to BS. High mobility of vehicles in expressway can cause the changes of vehicular network topology and forms the dynamic shortest transmission paths. Hence, the transmission rates may fluctuate. The BS is able to discover the current network topology in real time to establish the shortest transmission paths for vehicular nodes. The transmission rate is inversely proportional to the length of the transmission path. The system can derive the transmission rates in real time. It is assumed that the transmission rate does not change in each time slot. In addition, the forwarding mode of the BS can be considered as the other transmission path (source-BS-destination). The system will choose the path corresponding to the larger transmission rate, which is called the *selected* transmission path.

3.3. Task Model and Problem Formulation. When an edge server has an application request of ITS, it will create a task scheduling scheme. This type of request usually contains some information about the application type and relevant parameters. And then, the workflow tasks will be executed by the selected vehicular nodes according to the above scheduling scheme. A workflow application can be modeled as a directed acyclic graph $G = (V, E)$ in which V indicates the task set and elements in E represent the interdependency among tasks. If an edge (i, j) exists between task i and task j, it means that task i is a predecessor of task j. Meanwhile, edge (i, j) indicates that task j is a successor of task i. If a task has no predecessor, it is called an entry task. And a task which has no successor is called an exit task. The data size corresponding to each edge in a task graph is stored in an adjacency matrix. An entry task can be considered to have a virtual edge from itself to itself, and its input data size is also stored in the adjacency matrix. For an application request, the input data of the entry task will be transferred to the VC by the BS before task execution. The notations used in this paper are shown in Table 1.

Table 1: Notation definition.

Notation	Description
C_i	Computation load of task i
f_m	Computing speed of vehicle m
$d_{i,j}$	Data size corresponding to edge (i, j)
$b_{m,n}$	Transmission rate between vehicle m and vehicle n
$\text{pre}(i)$	The set of predecessors of task i
F	The set of exit tasks that have no successor
TS	Duration of one time slot
TD	Deadline of the requested application
$\text{TP}(i, m)$	Computing time of task i on vehicle m
$\text{TR}(i, j, m, n)$	Transmission time corresponding to $d_{i,j}$ and $b_{m,n}$
$\text{TA}(m)$	The earliest time at which vehicle m is ready for computing
$\text{EST}(i, m)$	The earliest start time of task i on vehicle m
$\text{EFT}(i, m)$	The earliest finish time of task i on vehicle m
$\text{AFT}(i, m)$	The actual finish time of task i on vehicle m
ACT	The application completion time, i.e., makespan

The computing time of task i on vehicle m can be calculated by the following equation:

$$\text{TP}(i, m) = \frac{C_i}{f_m}. \tag{1}$$

This paper interchangeably uses the two terms, computing time and execution time, and they all indicate the time that a CPU spends on the computation load of a task.

The transmission time of $d_{i,j}$ can be analyzed according to the dynamicity of the transmission rate. If the transmission rate does not change during the $d_{i,j}$ transmission, when the source node and destination node are vehicle m and vehicle n, respectively, the transmission time of $d_{i,j}$ can be given by

$$\text{TR}(i, j, m, n) = \frac{d_{i,j}}{b_{m,n}}. \tag{2}$$

If the transmission rate changes during the $d_{i,j}$ transmission, when the source node and destination node are vehicle m and vehicle n, respectively, the relationship between transmission time $\text{TR}(i, j, m, n)$ and date size $d_{i,j}$ can be given by

$$\sum_{t=1}^{\text{TR/TS}} b(m, n, t) \cdot \text{TS} = d_{i,j}, \tag{3}$$

where TR is the abbreviation of $\text{TR}(i, j, m, n)$ and $b_{(m,n,t)}$ is the transmission rate in timeslot t. For the above case, if the network topology changes frequently, the system will have a high computation cost when predicting all the transmission paths and the corresponding lifetimes.

The earliest start time of task i on vehicle m can be given by

$$\text{EST}(i, m) = \max\left\{\text{TA}(m), \max_{k\in\text{pre}(i)} (\text{AFT}(k) + \text{TR}(k, i))\right\}, \tag{4}$$

where $\text{TR}(k, i)$ is the transmission time of $d_{k,i}$ and $\text{AFT}(k)$ represents the actual finish time of task k. For task i, the corresponding earliest start time can be derived based on the finish time of its predecessors, input data transmission time, and the ready time of vehicle m. The earliest finish time of task i on vehicle m can be given by

$$\text{EFT}(i, m) = \text{TP}(i, m) + \text{EST}(i, m). \tag{5}$$

When task i is allocated to a vehicular node, the corresponding EFT is equal to its AFT.

If there is only one exit task in a given application, the makespan is the actual finish time of the exit task. Otherwise, the makespan is the time when all exit tasks are finished. Therefore, ACT (i.e., makespan) can be given by

$$\text{ACT} = \max_{i\in F}\{\text{AFT}(i)\}. \tag{6}$$

The objective of this paper is to minimize the makespan so as to improve the quality of VC services. The workflow scheduling problem in VC can be formulated as follows:

$$\min \max_{i\in F}\{\text{AFT}(i)\}. \tag{7}$$

4. The Proposed HDWS Strategy

HDWS contains a static scheduling module and dynamic scheduling module. HDWS first runs the static scheduling and then updates the schedule at each time slot using dynamic scheduling. The static scheduling contains steps (Section 4.1 to 4.5). The dynamic scheduling adjusts the schedule in real time according to dynamic transmission rates and execution progress.

4.1. Task Set Establishment. In order to control the makespan and reduce the complexity of the problem, HDWS partitions a given application to several task sets based on topological sorting. Then, the available time (i.e., from starting time to deadline) is divided into several time segments for the generated task sets.

Due to task dependencies in DAG, a task i must be executed after the executions of its predecessors. Therefore, it is unable to estimate the completion time of task i if the predecessors have not been totally scheduled. Task sets are established in the sequence of topological sorting so as to conform to scheduling sequence. For a DAG, each task that has the in-degree 0 will be added into a task set, and then, these tasks and the corresponding edges will be deleted from DAG. So, the in-degrees of the remaining tasks will be updated. Next, HDWS uses the above method to create a new task set composed of tasks with in-degree 0. Therefore, the task sets can

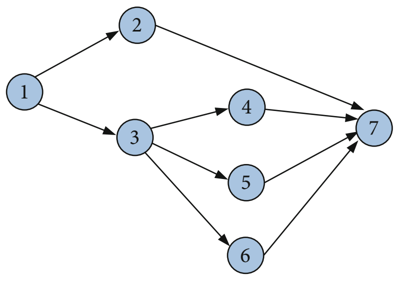


Figure 2: An example of the task graph.

be iteratively established. For example, all the tasks in Figure 2 can be partitioned into four sets sequentially, i.e., {1}, {2, 3}, {4, 5, 6}, and {7}.

4.2. Node Selection for the First Task Set. Tasks in the first task set need to download input data from the BS, so the data download rate is one factor of influencing the makespan. In order to alleviate the local optimum, we consider the following factors to select nodes for the first task set.

The first factor is computing speed. A node with faster computing speed can finish a task within less time. The second factor is the average computing speed of neighboring nodes. Considering that the successor tasks may be executed by several nodes in parallel, the neighboring nodes with stronger computing capability will be preferred. The number of neighboring nodes is the third factor. The last factor is the data download rate.

For each factor, a ranking is made to represent the corresponding superiority. HDWS adopts the sum of weighted ranking to quantify the advantages in scheduling. For node i, the four rankings are denoted as W_i, X_i, Y_i, Z_i and the weights are represented as α, β, γ, and δ, respectively. For α, β, γ, and δ, each parameter has a value in the range (0, 1). The selection priority of node i can be calculated by the following equation:

$$P_i = \alpha W_i + \beta X_i + \gamma Y_i + \delta Z_i. \tag{8}$$

The size of the first task set is denoted as q, and the top-q nodes in selection priority will be selected. Tasks in the first task set are sorted in a descending order by workload. The i th task is assigned to the ith node sorted by selection priority. If the number of nodes in VC is smaller than q, tasks will be assigned circularly to the existing nodes by the above method.

4.3. Time Partition. For the first task set, the estimated finish time can be derived by using the above approach and is denoted as T_1. The total number of task sets is denoted as m. So, $(\mathrm{TD} - T_1)$ is the rest available time and HDWS will divide it into $(m-1)$ time segments for the rest task sets. The available time is divided according to the particular workload of the rest task sets. For a task set, the finish time is usually dependent on the task with the maximal workload, because the tasks can be executed in parallel. Hence, HDWS uses the maximal workload as the particular workload for each task set. The workload of a task contains computation load and transmission load which have different attributes, so it is hard to directly quantify them together. To tackle this difficulty, the workload is transformed into working time that consists of transmission time and computing time. The workload of task i can be calculated by the following equation:

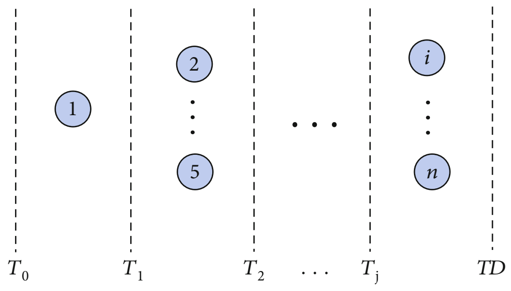


Figure 3: Time segment.

$$W_i = \frac{d_i}{b_v} + \frac{c_i}{f_v}, \tag{9}$$

where d_i denotes the size of input data of task i. Besides, b_v and f_v denote the maximal transmission rate and the maximal computing speed, respectively.

The particular workload of the jth task set S_j can be given by

$$W_p^j = \max_{i \in S_j} \{W_i\}, \quad j > 1. \tag{10}$$

The time segment for the task set S_j can be calculated by

$$\mathrm{TS}(j) = \frac{W_p^j}{\sum_{k=2}^{m} W_p^k} (\mathrm{TD} - T_1). \tag{11}$$

As shown in Figure 3, the duration $(\mathrm{TD} - T_1)$ can be partitioned into several time segments. The number of time segments is equal to the number of the rest task sets which do not contain the first task set. Hence, the time boundary can be derived by

$$T_j = \mathrm{TS}(j) + T_{j-1}, \tag{12}$$

and it is designed to control the finish time of task sets in order to satisfy the deadline constraint. Meanwhile, the time boundary is utilized to classify nodes to reduce problem complexity of local scheduling for each task set.

4.4. Node Classification and Local Scheduling. Compared to scheduling the entire workflow, it is easier to create a local scheduling for a task set to meet the requirement of the corresponding time boundary. For each task set, HDWS classifies nodes based on the time boundary to select suitable nodes as worker nodes. Hence, the number of worker nodes can be controlled. For a node m, the completion time of task i can be calculated by

```
Input: Number of tasks N, number of nodes M, current assignment ass, start time stime
Output: Node assignment
1: ACT = Caltime(ass, stime); //estimate the makespan
2: for each task set in the order of set establishment
3:    Sort tasks in ascending order by computation load;
4:    Add the sorted tasks into array tsort;
5: endfor
6: for each task i in tsort
7:    Sort all successors of task i in ascending order by computation load;
8:    Add the sorted successors into an array suc(i);
9:    Estimate the transmission rate between each node and ass[i];
10:   Sort all nodes in descending order by the above transmission rate;
11:   Add the sorted nodes into an array veh(i);
12:   for each node j in veh(i)
13:       for each task k in suc(i)
14:           reass = ass;
15:           reass[k] = j;
16:           newtime = Caltime(reass, stime);
17:           if (newtime < ACT).
18:               ACT = newtime;
19:               ass[k] = reass[k];
20:           endif
21:       endfor
22:   endfor
23: endfor
```

Algorithm 1: Successor Task Reassignment.

```
Input: Number of tasks N, number of nodes M, current assignment ass, start time stime
Output: Node assignment
1: ACT = Caltime(ass, stime); //estimate the makespan
2: for each task set in the order of set establishment
3:    Sort tasks in ascending order by computation load;
4:    Add the sorted tasks into array tsort;
5: endfor
6: for each task i in tsort
7:    Add all successors of task i into a set suc(i);
8:    for each node j
9:      sum_rate =0;
10:       for each task k in suc(i)
11:           sum_rate = sum_rate + b(j, ass[k]);
12:       endfor
13:       ave_rate(j) = sum_rate / |suc(i)|;
14:   endfor
15:   Sort all nodes in descending order by ave_rate;
16:   Add the sorted nodes into an array veh(i);
17:   for each node j in veh(i)
18:       reass = ass;
19:       reass[i] = j;
20:       newtime = Caltime(reass, stime);
21:       if (newtime < ACT)
22:           ACT= newtime;
23:           ass[i] = reass[i];
24:       endif
25:   endfor
26: endfor
```

Algorithm 2: Predecessor Task Reassignment.

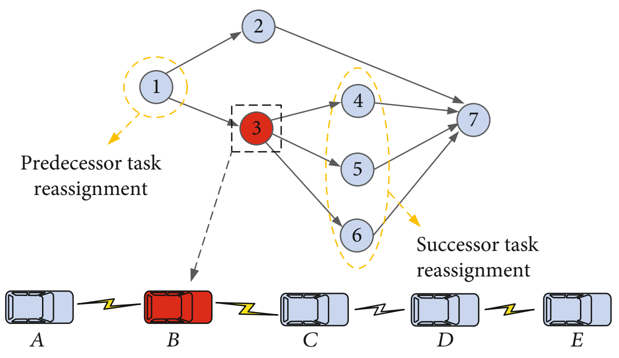


FIGURE 4: Task reassignment.

$$\mathrm{CT}(i,m)=\max\{\mathrm{AFT}(k)+\mathrm{TR}(k,i)+\mathrm{TP}(i,m)\},\quad k\in \mathrm{pre}(i). \tag{13}$$

Since the scheduling of the previous task set has been made, $\mathrm{TR}(k,i)$ can be derived.

For task set S_j, the nodes which satisfy the following inequalities will be selected as worker nodes to execute tasks.

$$\begin{aligned} CT(i,m)&\le T_j,\quad i\in S_j,\\ \mathrm{POS}(m,\mathrm{CT}(i,m))&\le L,\quad i\in S_j, \end{aligned} \tag{14}$$

where $\mathrm{POS}(m,t)$ is the updated position of node m after a period of time t and L represents the boundary of VC. The other nodes are referred to as alternative nodes. If the number of worker nodes is smaller than the number of tasks in the task set, alternative nodes will be added into the worker node set so that their amounts are equal. This operation can improve the parallelization of task execution.

Local scheduling is performed iteratively in the sequence of task set establishment except the first task set. The specific steps are as follows. If the number of tasks in a task set is larger than h, local scheduling will sort tasks in a descending order by workload and assign the top-h tasks at first. For the top-h tasks, local scheduling traverses all mapping between tasks and the worker nodes to find the local optimal scheduling which minimizes their maximal completion time. For the rest tasks in the current task set, local scheduling performs the above operations iteratively until all the rest tasks have been assigned. Otherwise, all the tasks in the task set will be assigned by the local optimal scheduling. The value of h will be reasonably set to guarantee the acceptable computation complexity.

4.5. Task Reassignment. The successor task reassignment (STR) and predecessor task reassignment (PTR) algorithms are proposed in order to further reduce ACT-based on local scheduling. The core idea of these two algorithms is to reassign tasks to nodes corresponding to higher transmission rates to search better solution with acceptable time complexity.

The STR algorithm mainly contains three phases: successor sorting phase, node sorting phase, and node selection phase.

4.5.1. Successor Sorting Phase. At first, tasks in each task set are sorted independently in an ascending order by computation load and the sorted tasks are added into an array $t$sort. For each task in $t$sort, its successors are sorted in an ascending order by computation load.

4.5.2. Node Sorting Phase. This phase estimates the transmission rates between each node and already designated node of each task i (in $t$sort) whose successors will be reassigned to reduce ACT. All nodes are sorted in a descending order by the above transmission rate. The higher transmission rate can reduce data transmission time. If the designated node of the successor is the same with task i, the data transmission time is zero.

4.5.3. Node Selection Phase. According to successor sorting and node sorting, successor task reassignment is iteratively performed to retain the scheduling result of less ACT. In addition, when STR tries to evaluate a candidate node, the position of the node will be calculated according to the updated task finish time. If the updated position is out of the coverage of BS, the node will not be selected.

The pseudocode of the STR algorithm is shown in Algorithm 1.

The STR algorithm has an $O(vqe^2)$ time complexity for v tasks, q nodes, and e edges.

The PTR algorithm has three phases: predecessor sorting phase, node sorting phase, and node selection phase.

4.5.4. Predecessor Sorting Phase. Tasks in each task set are sorted independently in an ascending order by computation load, and the sorted tasks are added into an array $t$sort.

4.5.5. Node Sorting Phase. This phase estimates the average transmission rates between each node and already assigned nodes of successors of each task i in $t$sort (viewed as predecessors) which will be reassigned to reduce ACT. All nodes are sorted in a descending order by the average transmission rate.

4.5.6. Node Selection Phase. According to predecessor sorting and node sorting, predecessor task reassignment is iteratively performed to retain the scheduling result of less ACT . In addition, when PTR tries to evaluate a candidate node, the position of the node will be also updated to check whether it satisfies the system requirement.

The pseudocode of the PTR algorithm is shown in Algorithm 2.

For v tasks, q nodes and e edges, the PTR algorithm has an $O(vq(q+e))$ time complexity.

As an illustration, Figure 4 presents the basic explanation of STR and PTR algorithms. Specifically, when task 3 has been assigned to node B, its successors $\{4,5,6\}$ will be reassigned to other nodes which have an advantage in the transmission rate by the STR algorithm and the better solution will be retained. For the PTR algorithm, the predecessor of task 3 (i.e., task 1) will be reassigned and the average transmission rate corresponding to edge $(1,2)$ and $(1,3)$ is used to search a more suitable node in order to achieve less makespan.

Table 2: Parameters of VC and workflow.

Parameters	Value
The coverage range of BS	1000 m
The communication range between vehicles	250 m
Computing speed of the vehicle	[1000, 1500] MHz
Duration of one time slot	0.01 s
Moving speed of the vehicle	[100, 120] km/h
Download rates of different road segments	{3, 6, 12, 24} Mbps
Data size of workflows a, b, c, and d	(1510, 3210, 3600, 2400) KB
The minimum of the computation load of one task	200 Mcycle
The maximum of the computation load of one task	2900 Mcycle

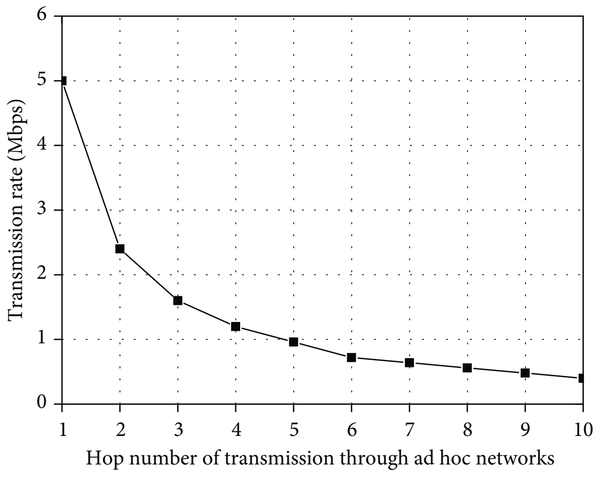


Figure 5: Transmission rate.

4.6. Dynamic Scheduling. The mobility of vehicular nodes leads to dynamic network topology which causes the change of the transmission rate between the source node and the destination node. However, static task scheduling derives transmission rates and makes decisions based on the static network topology, which is not suitable for the dynamic scene. To tackle this problem, a dynamic scheduling scheme is designed to adjust node assignment according to network topology.

The core idea is to discover the changes of transmission rates in real time and determine the time to trigger the scheduling scheme again for unready tasks based on the updated transmission rates. In each time slot, HDWS estimates and records the transmission rates for all pairs of nodes based on the current network topology. If there is any change of transmission rates compared to the previous time slot, HDWS will make decision according to the specific change.

All the situations of changes of transmission rates are classified into three categories, and the detailed operations are described as follows:

(1) Situation: transmission rates change and the current data transmissions of one or more tasks are affected

  (i) Estimate the finish time of affected tasks based on the current transmission rates and transmission status. Estimate the finish time of tasks which are ready to execute or in execution status. The maximal value of the above time is denoted as *maxpoint*

  (ii) Estimate the transmission start time of tasks which have not started to receive input data. The minimal value of the above time is denoted as *minpoint*

  (iii) If *minpoint* is equal to or greater than *maxpoint*, trigger the above scheduling scheme (Section 4.4–4.5) at time *maxpoint* for the unready tasks based on transmission rates corresponding to *maxpoint* (denoted as operation A)

  (iv) Otherwise, the system continues to run based on the original scheduling (denoted as operation B)

(2) Situation: transmission rates change, and there is no task whose current data transmission is affected

  (i) Estimate the finish time of tasks which are in transmission status. The maximal value of the above time is denoted as *maxpoint*

  (ii) Estimate the transmission start time of tasks which have not started to receive input data. The minimal value of the above time is denoted as *minpoint*

  (iii) If *minpoint* is equal to or greater than *maxpoint*, perform operation A

  (iv) Otherwise, perform operation B

(3) Situation: transmission rates change, and there is no data transmission in the current time slot

  (i) Estimate the finish time of tasks which are ready to execute or in execution status. The maximal value of the above time is denoted as *maxpoint*

  (ii) Estimate the transmission start time of tasks which have not started to receive input data. The minimal value of the above time is denoted as *minpoint*

  (iii) If *minpoint* is equal to or greater than *maxpoint*, perform operation A

  (iv) Otherwise, perform operation B

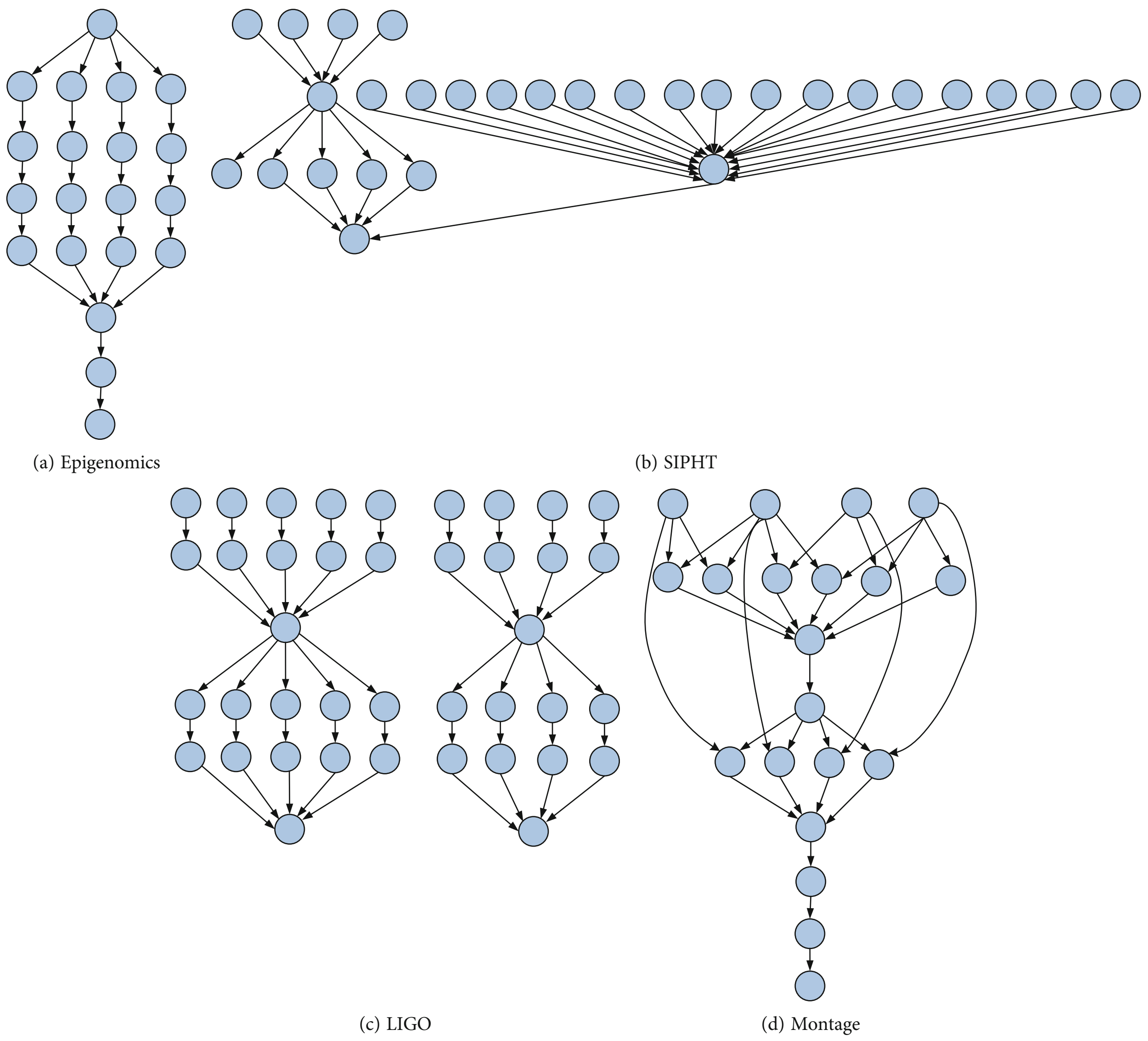


Figure 6: Workflow test instances.

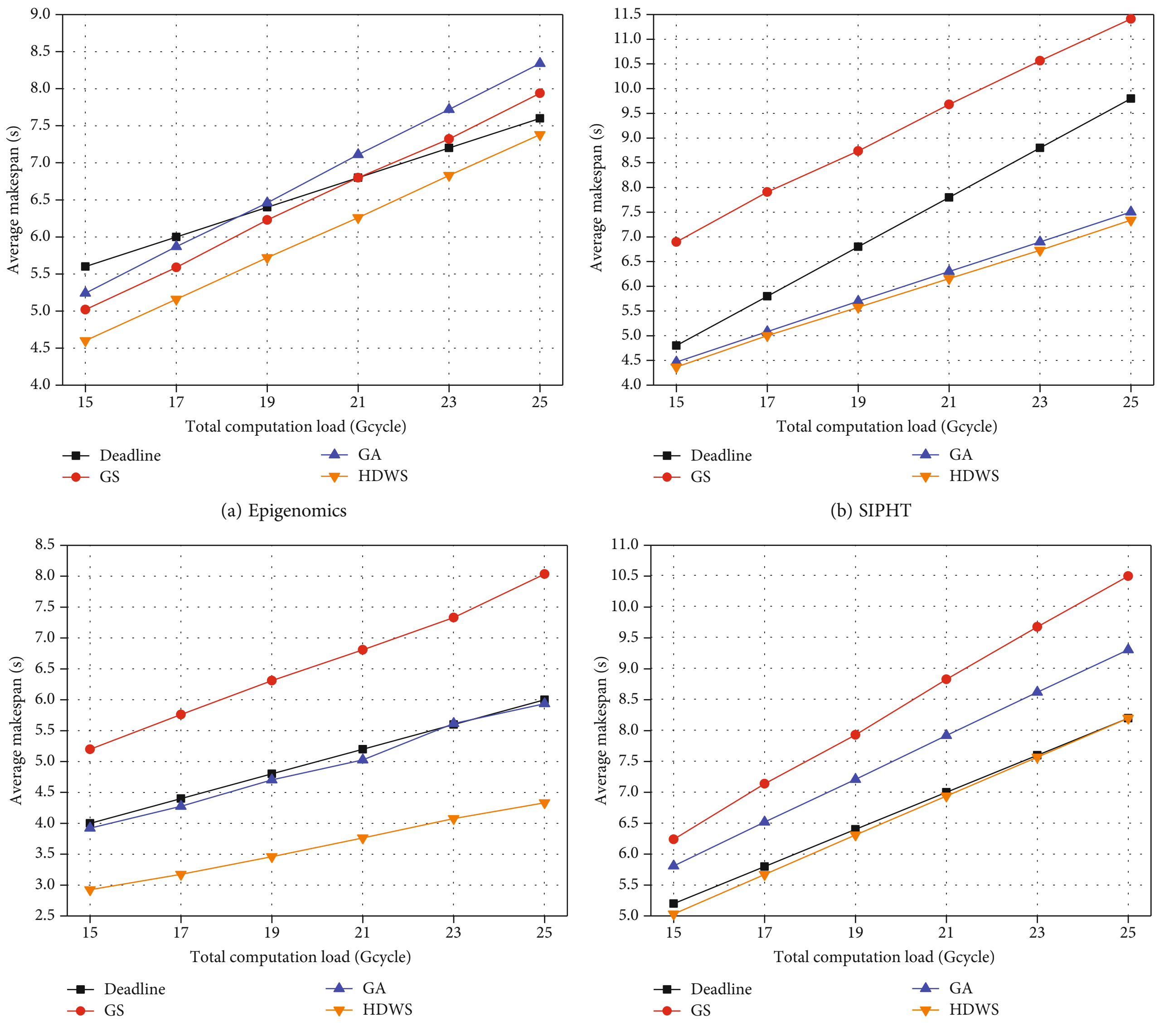
5. Performance Evaluation

In this section, the performance evaluation on the HDWS strategy and two existing methods is presented. The performance metrics contain the service success ratio and average makespan. A requested application has a deadline. If the makespan does not exceed the deadline, it is considered that the service is successful. The metrics of the service success ratio and makespan reflect service stability and service efficiency of VC, respectively.

A simulator is designed to simulate the dynamic multihop transmissions among vehicles and all the execution processes of VC. In the simulator, a counter is utilized to emulate and record time sequences. When a time slot passes, the value of the counter will be increased by one. The simulator is implemented by C++, and it is able to record the system state at each time slot. The simulation runs 100 times for each workflow instance, and the initial positions of nodes are randomly generated in the interval of [0, 520] m each time.

5.1. Experimental Setup. In this section, the relevant setting and parameters in simulation experiment is described from the following aspects.

5.1.1. Vehicular Cloud. VC services in expressway are simulated, and vehicles move at constant speeds. The parameters about VC are described in Table 2. The transmission rate between a source node and a destination node through ad hoc networks is set to the value in Figure 5. Vehicles on different road segments have different download rates from BS. The total service range is divided into seven road segments which have lengths of 200, 100, 100, 200, 100, 100, and 200 (m). So the fourth road segment corresponds to the highest download rate, since it is closer to BS compared with other road segments. Since downlink and uplink are asymmetric, the upload rate to BS is usually much smaller than the download rate. The transmission rate corresponding to the mode of BS forwarding is set to 0.2 Mbps.



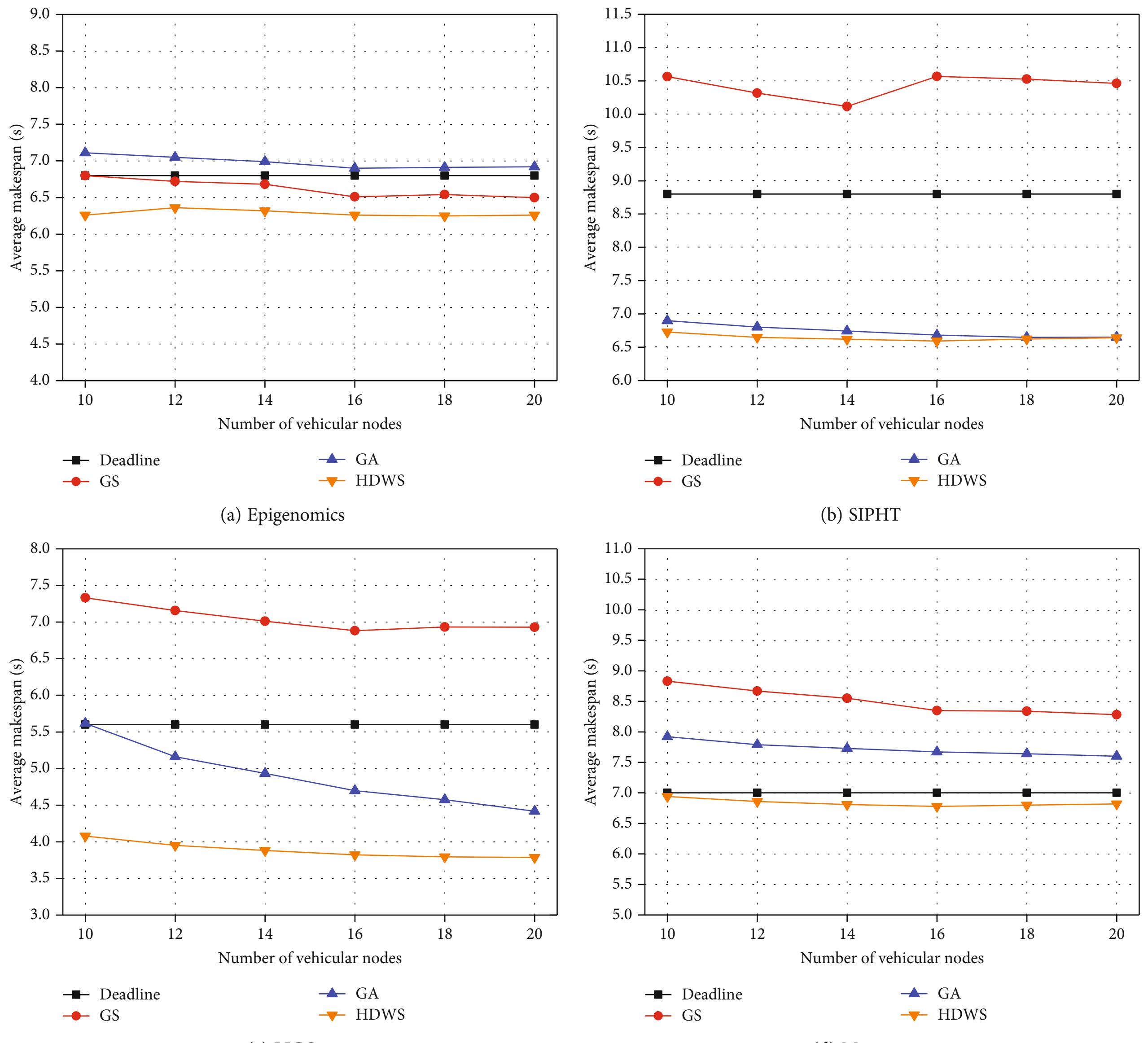
(a) Epigenomics (b) SIPHT (c) LIGO (d) Montage

Figure 7: Average makespan with respect to the total computation load.

5.1.2. Workflow Application. Four standardized workflows Epigenomics [26], SIPHT [27], LIGO [28], and Montage [29] with different workloads are used to evaluate performances. The specific task graphs are shown in Figure 6, and the specific parameters are illustrated in Table 2.

5.1.3. Scheduling Scheme. The parameter h is set to 6. The sum of α, β, γ, and δ is one, and they are set to 0.5, 0.1, 0.1, and 0.3, respectively. Genetic algorithm (GA) was modified to perform as a workflow scheduling algorithm called MGA in VC [6]. Since MGA only considers the fixed transmission time among tasks, which is not suitable for dynamic transmission, hence, we select GA rather than MGA as the comparison scheme. As far as we know, there is no research on workflow scheduling strategy considering dynamic transmission rates by now. So, greedy scheduling (GS) as a baseline approach is also selected for comparison. GS sorts tasks in the sequence of topological order and allocates the node with the least task completion time to the current task. In GA, the iteration number of genetic operations is set to 200 and the number of individuals is set to 10. A larger iteration number such as 500 will exceed the duration of one time slot and violate the requirement of being real time of the application.

5.2. Experimental Results and Analysis. Figure 7 shows the average makespan of GA, GS, and the proposed HDWS with different computation loads. The number of vehicular nodes is set to 10 for each workflow. When the data size of each workflow remains unchanged, the average makespan of GA, GS, and HDWS shows an increasing trend with the increase of the total computation load. As the total computation load increases, the more execution time will be taken so that the corresponding makespan is increased. HDWS has less average makespan than comparison schemes and satisfies the corresponding deadline for all tested workflows.



(a) Epigenomics

(b) SIPHT

(c) LIGO

(d) Montage

Figure 8: Average makespan with respect to the number of vehicular nodes.

For workflow Epigenomics, the average makespan of HDWS is 7% and 11% less than GS and GA, respectively, and GA does not satisfy the deadline when the total computation load is not less than 19 Gcycle. For workflow SIPHT, the average makespan of HDWS is 36% less than GS and it is a little superior to GA. For workflow LIGO, the average makespan of HDWS is 44% and 26% less than GS and GA, respectively, and GS does not satisfy the deadline for all tested computation loads. For workflow Montage, the average makespan of HDWS is 20% and 12% less than GS and GA, respectively, and GS does not satisfy the deadline for all tested computation loads as well as GA.

The number of vehicular nodes overall reflects the amount of computation resource in VC. In order to evaluate the impact of the number of vehicular nodes on the average makespan, we fix the computation load, data size, and deadline for each workflow. The total computation load for workflow Epigenomics is set to 21 Gcycle as well as Montage. The total computation load for workflow SIPHT is set to 23 Gcycle as well as LIGO. The number of vehicular nodes is set to 10, 12, 14, 16, 18, and 20. As shown in Figure 8, the average makespan of GA, GS, and HDWS shows a little decreasing trend with the increase of the number of vehicular nodes in general. As the number of vehicular nodes increases, the scheduling schemes utilize more computation resources to speed up execution. HDWS achieves less average makespan than GS and GA for all tested workflows with respect to the number of nodes. For workflow Epigenomics, the average makespan of HDWS is 5% and 9% less than GS and GA, respectively, and GA does not satisfy the deadline for all tested number of nodes. For workflow SIPHT, the average makespan of HDWS is 36% less than GS and it is a little superior to GA. For workflow LIGO, the average makespan of HDWS is 44% and 20% less than GS and

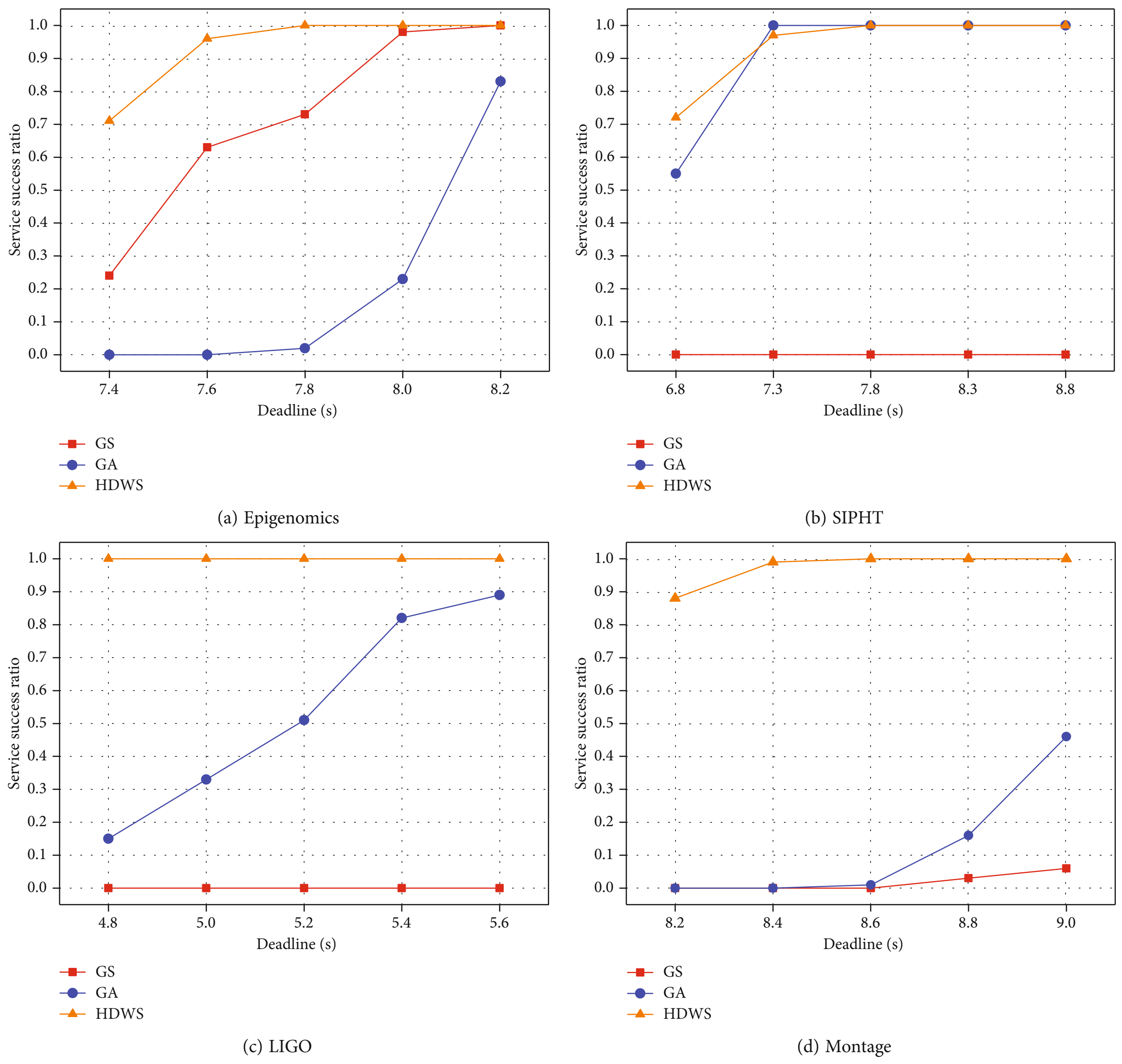


Figure 9: Service success ratio with respect to deadline.

GA, respectively, and GS does not satisfy the deadline for all tested number of nodes. For workflow Montage, the average makespan of HDWS is 19% and 11% less than GS and GA, respectively, and GA does not satisfy the deadline for all tested number of nodes as well as GS.

As shown in Figure 9, the service success ratio (SSR) with respect to deadline is evaluated. Deadline has a direct impact on SSR and represents the required real time of an application. For workflow Epigenomics and Montage, the number of vehicular nodes is set to 16 and the total computation load is set to 25 Gcycle. For workflow SIPHT and LIGO, the number of vehicular nodes is set to 12 and the total computation load is set to 23 Gcycle. The SSR of GA, GS, and HDWS shows an nondecreasing trend with the increase of the deadline. For workflow Epigenomics, the SSR of HDWS is averagely higher than GS and GA by 26% and 71%, respectively. For workflow SIPHT, the SSR of HDWS is averagely higher than GA by 2% and GS has no successful service for all tested deadlines. For workflow LIGO, the SSR of HDWS is averagely higher than GA by 46% and GS has no successful service for all tested deadlines. For workflow Montage, the SSR of HDWS is averagely higher than GS and GA by 94% and 83%, respectively, and GS has no successful service when the deadline is not greater than 8.4 s as well as GA.

6. Conclusion

In this paper, the problem of workflow scheduling in VC under the condition of dynamic multihop transmission is studied. We propose a dynamic scheduling strategy called HDWS which contains task assignment algorithms and dynamic scheduling scheme. HDWS utilizes the current system status in real time to complete dynamic scheduling to cope with dynamic transmission rates. The experimental results show that the HDWS strategy improves efficiency and stability of VC services. In our future work, energy consumption of smart vehicles in vehicular cloud computing will be considered. Hence, energy-conserved task scheduling will be investigated in vehicular cloud. The scheduling of task execution and data transmission will be improved to make a good tradeoff between energy consumption and quality of services.

Data Availability

The datasets generated during the current study are available from the corresponding author upon reasonable request.

Conflicts of Interest

The author declares that there are no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (grant number 61802181).

References

[1] S. K. Pande, S. K. Panda, S. Das et al., "A smart cloud service management algorithm for vehicular clouds," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 5329–5340, 2021.

[2] M. Eltoweissy, S. Olariu, and M. Younis, "Towards autonomous vehicular clouds," in *Ad Hoc Networks. ADHOCNETS 2010*, J. Zheng, D. Simplot-Ryl, and V. C. M. Leung, Eds., vol. 49 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pp. 1–16, Springer, Berlin, Heidelberg, 2010.

[3] Y. Fei, M. Adams, and S. Roy, "V2V wireless communication protocol for rear-end collision avoidance on highways," in *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, pp. 375–379, Beijing, China, 2008.

[4] J. Lee and B. Park, "Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 1, pp. 81–90, 2012.

[5] S. Demmel, D. Gruyer, and A. Rakotonirainy, "V2V/V2I augmented maps: state-of-the-art and contribution to real-time crash risk assessment," in *Proceedings of the 20th Canadian Multidisciplinary Road Safety Conference*, pp. 1–15, Niagara Falls, Ontario, Canada, 2010.

[6] F. Sun, F. Hou, N. Cheng et al., "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11049–11061, 2018.

[7] I. Sorkhoh, D. Ebrahimi, R. Atallah, and C. Assi, "Workload scheduling in vehicular networks with edge cloud capabilities," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 8472–8486, 2019.

[8] S. Olariu, "A survey of vehicular cloud research: trends, applications and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 6, pp. 2648–2663, 2020.

[9] P. Li, X. Chen, and S. Shen, "Stereo r-cnn based 3d object detection for autonomous driving," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7644–7652, Long Beach, CA, USA, 2019.

[10] X. Ma, Z. Wang, H. Li, P. Zhang, W. Ouyang, and X. Fan, "Accurate monocular 3d object detection via color-embedded 3d reconstruction for autonomous driving," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*pp. 6851–6860, In, Seoul, Korea (South), 2019.

[11] J. Ullman, "*NP*-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[12] R. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1787–1796, 2014.

[13] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multiobjective workflow scheduling in cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1344–1357, 2016.

[14] P. Kaur and S. Mehta, "Resource provisioning and work flow scheduling in clouds using augmented shuffled frog leaping algorithm," *Journal of Parallel and Distributed Computing*, vol. 101, pp. 41–50, 2017.

[15] H. Wu, X. Hua, Z. Li, and S. Ren, "Resource and instance hour minimization for deadline constrained DAG applications using computer clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 885–899, 2016.

[16] J. Sahni and D. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 2–18, 2018.

[17] Q. Qi, J. Wang, Z. Ma et al., "Knowledge-driven service offloading decision for vehicular edge computing: a deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4192–4203, 2019.

[18] Y. Liu, S. Wang, Q. Zhao et al., "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4961–4971, 2020.

[19] Y. Ku, S. Baidya, and S. Dey, "Adaptive computation partitioning and offloading in real-time sustainable vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13221–13237, 2021.

[20] L. Liu, H. Tan, S. Jiang, Z. Han, X. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *IWQoS '19: Proceedings of the International Symposium on Quality of Service*, pp. 1–10, Phoenix, AZ, USA, 2019.

[21] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, Toronto, ON, Canada, 2020.

[22] J. Yan, S. Bi, and Y. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: a deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5404–5419, 2020.

[23] S. K. Panda and P. K. Jana, "Uncertainty-based QoS min-min algorithm for heterogeneous multi-cloud environment," *Arabian Journal for Science and Engineering*, vol. 41, no. 8, pp. 3003–3025, 2016.

[24] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-aware online scheduling for real-time workflows in cloud service

environment," *IEEE Transactions on Services Computing*, vol. 14, no. 4, pp. 1167–1178, 2021.

[25] S. K. Pande, S. K. Panda, and S. Das, "Dynamic service migration and resource management for vehicular clouds," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 1, pp. 1227–1247, 2021.

[26] "Illumina," April 2015, https://www.illumina.com/.

[27] J. Livny, H. Teonadi, M. Livny, and M. K. Waldor, "High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs," *PLoS One*, vol. 3, no. 9, article e3197, 2008.

[28] D. A. Brown, P. R. Brady, A. Dietz et al., "A case study on the use of workflow technologies for scientific analysis: gravitationalwave data analysis," in *Workflows for E-Science*, I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, Eds., pp. 39–59, Springer, London, 2007.

[29] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the Montage example," in *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, Austin, TX, USA, 2008.