Cost-Effective Task Scheduling Strategy in Vehicular Cloud Computing

Jie Wang

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

email: jieb@nuaa.edu.cn

Qiang Zhang *

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

email: cszhangqiang@nuaa.edu.cn

ABSTRACT

With the development of vehicle cloud computing technology, vehicle cloud has more extensive application scenarios. It can provide users with computing resources to process requests of mobile devices, so as to improve the defect of weak computing and storage capacity of mobile devices. Due to the distributed resources of vehicle cloud, it is a challenge to schedule workflow tasks with deadline constraint and high cost efficiency. In this paper, we propose a cost-effective and deadline-constrained strategy (CEDCS) to implement task scheduling in vehicular cloud computing. Our strategy firstly proposes an initial task allocation strategy to create a schedule to meet deadline constraint. Then, a task rescheduling algorithm is proposed to achieve lower service cost. Our strategy is compared with existing genetic algorithm (GA) through simulation experiments. Experimental results show that our strategy can achieve higher service success ratio and lower service cost.

CCS CONCEPTS

•**Networks~Network services~Cloud computing**

KEYWORDS

Vehicular Cloud Computing, Task Scheduling, Service Cost, Deadline

ACM Reference Format:

1 INTRODUCTION

With the development of computer technology and communication technology, mobile terminal is playing a more and more important role in people's daily life. Mobile terminals such as mobile phones, tablets and watches provide a variety of services, such as entertainment, life and office, which not only provide convenience, but also provide functions that other tools cannot provide. On the other hand, mobile applications have brought great challenges to the computing capacity, storage capacity and battery capacity of mobile terminals [1]. Compared with large computers, mobile devices have many disadvantages such as weak task processing ability, insufficient memory and poor battery life. When

* Corresponding author.

compute-intensive or data-intensive requests need to be processed, problems such as poor computing performance, high latency, and downtime occur on mobile devices, which seriously affects user's experience. With the increasing number of vehicles and the gradual enhancement of functions, these vehicles with strong data transmission capacity and sufficient storage space can be combined to form vehicle cloud to process user's requests [2]. Vehicle cloud computing (VCC) [3, 4] is a new computing model that combines vehicle network and mobile cloud computing. The emergence of VCC solves the problem of limited capabilities of mobile devices. By transmitting large applications that cannot be processed by mobile devices to vehicle cloud with powerful computing capabilities [5], the burden of mobile devices is reduced and user's experience is improved.

In recent years, VCC has attracted extensive attention and discussion in academic circles. Using VCC to complete user's requests can improve the processing capacity of large tasks and effectively reduce application completion time. However, due to the high speed of vehicles, vehicles as resource nodes of vehicle cloud are unstable, and vehicle network is often disconnected. Therefore, how to design an efficient task scheduling algorithm to meet the requirement of application deadline and meanwhile reduce service cost of vehicular cloud is a great challenge [6]. We propose a cost-effective and deadline-constrained strategy (CEDCS) for vehicular cloud services, to decide which task of an application to be served by which vehicular node. Our strategy can effectively improve the service success ratio and reduce the service cost. The main contributions of this paper are as follows:

1. Existing studies do not comprehensively consider the complete process of task transmission and processing, or do not take service cost as the optimization goal under the constraint of deadline. In this paper, we discuss how to schedule workflow tasks in vehicle cloud with our goal. In this paper, we also consider the mobility of vehicle cloud and the complex dependency of tasks.
2. This paper designs task scheduling strategy with two phases: initial task allocation and task rescheduling. The initial allocation solution fully considers the characteristics of cloud environment and tasks and meets the deadline. Then, based on the initial allocation strategy, a rescheduling algorithm is proposed to reduce service cost. Simulation results show that the proposed scheduling strategy has good performance.

The remainder of our paper is organized as follows. The second part describes the system model. The third part describes the system model. The fourth part introduces the proposed scheduling strategy. The fifth part is performance evaluation. The sixth part concludes this paper.

2 RELATED WORK

Task scheduling is always an important problem in distributed systems. For a bounded number of heterogeneous processors with an objective to simultaneously meet high performance and fast scheduling time, [7] proposes two scheduling algorithms called HEFT and CPOP. HEFT selects the task with the highest upward rank value at each step and assigns the selected task to the processor, which minimizes its earliest finish time with an insertion-based approach. CPOP uses the summation of upward and downward rank values for prioritizing tasks, and schedules the critical tasks onto the processor that minimizes the total execution time of the critical tasks in the processor selection phase.

[8] proposes a resource provisioning mechanism and a workflow scheduling algorithm, named GRP-HEFT, for minimizing the makespan of a given workflow subject to a budget constraint for the hourly-based cost model of modern IaaS clouds. It proposes a greedy algorithm which lists the instance types according to their efficiency rate.

[9] develops a performance modeling and prediction method for independent microservices, design a three-layer performance model for microservice-based applications. To minimize the end-to-end delay under a user-specified budget

constraint, it proposes a scheduling algorithm called GRCP. GRCP recursively selects and maps the functions both on the critical path and on the non-critical paths using a greedy procedure. It then refines the generated mapping scheme by minimizing the performance degradation if the budget constraint is violated.

[10] proposes a meta-heuristic cost effective genetic algorithm that minimizes the execution cost of the workflow while meeting the deadline in cloud computing environment. It develops schemes for encoding, population initialization, crossover, and mutations operators of genetic algorithm. It considers the essential characteristics of the cloud as well as VM performance variation and acquisition delay.

[11] studys the problem of computation offloading through the vehicular cloud, where computing missions from edge cloud can be offloaded and executed cooperatively by vehicles. For low complexity, a modified genetic algorithm based scheduling scheme is designed where integer coding is used rather than binary coding, and relatives are defined and employed to avoid infeasible solutions.

3 SYSTEM MODEL

3.1 Task Model

Applications that user need to process through VCC can be represented as task workflows with priority constraints [12], represented by directed acyclic graph (DAG) with G=(T, E), as shown in Figure 1. $T=\{t_1,t_2,...,t_k\}$ represents tasks, where K represents the number of tasks. $E \in T \times T$ indicates a dependency between tasks. If an edge $e_{i,j}=(t_i, t_j) \in E$ exists, t_j can only be executed after t_i completes, and t_i is a precursor and t_j is a successor. Precursors of t_i can be expressed as $Prec(t_i)=\{t_p|\exists e_{p,i} \in E\}$, successors of t_i can be expressed as $Succ(t_i)=\{t_s|\exists e_{i,s} \in E\}$. A task can be executed only after all its precursors have been executed. If a task does not have a precursor, it is called an entry task t_{entry}, which is $Prec(t_{entry})=\emptyset$. If a task has no successor, it is called an exit task t_{exit} which is $Succ(t_{exit})=\emptyset$. To ensure generality, we assumes that a DAG has only one entry task and one exit task. The computation load of t_i is represented by $L_p(t_i)$, and the transmission load is represented by $L_c(t_i)$. Task dependence matrix $I=(I_{i,j})_{K*K}$ represents the dependency between tasks and is the adjacency matrix of DAG [13], where $I_{i,j}>0$ indicates that t_i is the precursor of t_j, and $I_{i,j}$ is the value of is the transmission load of t_i. $I_{i,j}=0$ indicates that there is no dependency between t_i and t_j.

A DAG consisting of 21 tasks with dependencies is shown in Figure 1:

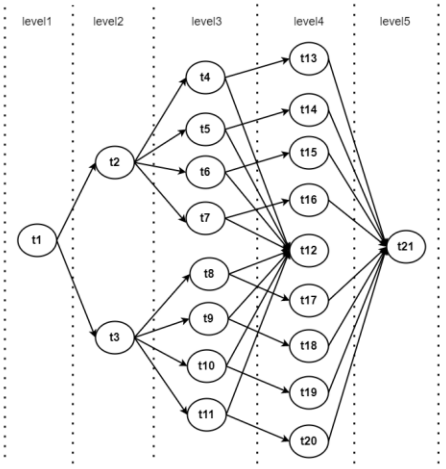

Figure 1: DAG

3.2 Vehicle Cloud System Model

Task execution process of vehicle cloud system is shown in Figure 2:

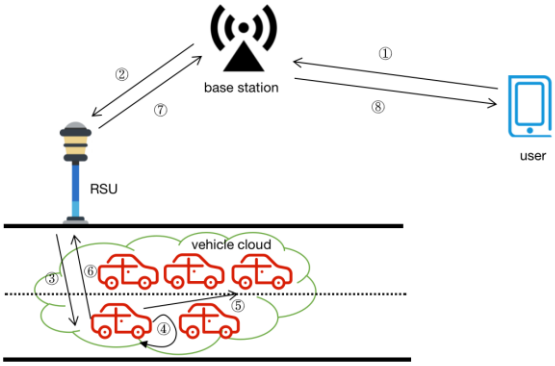

Figure 2: Task execution process of vehicle cloud system

As can be seen from Figure 2, many vehicles are driving on a section of road, and the movement of these vehicles can be regarded as on the same plane. As resource nodes with computation and transmission capabilities, vehicles are used to process user's requests to reduce the pressure on mobile terminals. $V=\{v_1,v_2,...,v_m\}$ represents vehicles, where M represents the number of vehicles. Due to the difference of CPU computation capabilities of vehicle chip and storage capacity of vehicle memory, vehicles have different task computation capabilities. According to the different task computation capabilities of vehicles, the service cost can be calculated based on the task computation time. The computation capability of v_i is $Cap(v_i)$, and the unit price of v_i is $UP(v_i)$. Different vehicle nodes have different unit prices. In this article, one task can only be served by one node, and cooperation between nodes is not considered to complete the task. After a node finishes the task computation, the task is transmitted to the next node through the wireless network. The transmission rate between v_i and v_j is $Rate(v_{i,j})$, which is determined by the physical location information of two vehicles and the network topology of vehicle nodes.

Road side unit (RSU) is installed on the side of the road as the communication center between the user, base station and vehicle nodes. RSU is responsible for data interaction with vehicle nodes and transmit the input data of request to the vehicle cloud. Considering the mobility of vehicles, location information of vehicles and network topology information of vehicle nodes are collected and saved by RSU. Vehicles within the RSU communication range constitute the vehicle cloud. Once a vehicle leaves the RSU range, it will not be able to complete the transmission and computation of user's tasks. Interval indicates the available time interval of a node in vehicular cloud.

The computation time of t_i on the vehicle node is denoted as $T_p(t_i)$, and the transmission time of t_i between two nodes is denoted as $T_c(t_i)$.

$T_p(t_i)$ is the ratio of the computation load to the computation capability of the node. $T_p(t_i)$ is formulated as:

$$T_p(t_i)=\frac{L_p(t_i)}{Cap(v_i)} \tag{1}$$

Similarly, $T_c(t_i)$ is the ratio of the transmission load to the transmission rate. $T_c(t_i)$ is formulated as:

$$T_c(t_i)=\frac{L_c(t_i)}{Rate(v_{i,j})} \tag{2}$$

Considering that user's requests need to be transmitted through interaction with RSU before they are processed in the vehicle cloud, the interaction time of user's requests between RSU and vehicle cloud is represented as InputRSU.

InputRSU can be regarded as a transmission time required by t_{entry} before computation, and the size of InputRSU is related to road conditions.

A task cannot begin processing until all its precursors have completed data transmission. The time at which all precursors have completed data transmission is called earliest start time (EST). $EST(t_i)$ is formulated as:

$$EST(t_i)=\begin{cases} InputRSU & ,if\ t_i = t_{entry} \\ \max_{t_p \in Pred(t_i)}\{EST(t_p) + T_p(t_p) + T_c(t_p)\} & ,otherwise \end{cases} \quad (3)$$

After data transmission, the task is transmitted to the vehicle node assigned in the task scheduling scheme and processed to reach the task completion state, which is called finish time (FT). $FT(t_i)$ is formulated as:

$$FT(t_i)=EST(t_i)+T_p(t_i) \quad (4)$$

t_{exit} in the application is the last task to be executed and the task with the longest FT among all tasks. When t_{exit} is completed, the entire application is completed, and the finish time of t_{exit} is the total finish time (TFT). TFT is formulated as:

$$TFT=FT(t_{exit}) \quad (5)$$

Vehicle cloud service providers provide VCC services and consume certain service cost. The service cost consumed by the execution of each task is related to the computation time of the task on the assigned node and the unit price of the node. $Cost(t_i)$ is formulated as:

$$Cost(t_i)=T_p(t_i)*UP(v_i) \quad (6)$$

The total cost (TC) for computing all tasks in the DAG can be obtained by summing up the service cost incurred by each task. TC is formulated as:

$$TC=\sum_{i=1}^{K} Cost(t_i) \quad (7)$$

3.3 Problem Definition

When a group of interdependent tasks make requests to the vehicle cloud, it is necessary to determine the mapping relationship between tasks and vehicle nodes, as well as the task execution sequence on the nodes, so as to keep TC as low as possible. As user have certain requirement on the real-time performance of tasks, a deadline (D) [14] is given to the application requesting the vehicle cloud. As the maximum application finish time acceptable to user, the application is considered successful only when it is completed within the deadline.

In this paper, the task scheduling problem is studied to meet the user-defined deadline and pursue the lowest possible service cost. According to Formula 5 and 7, the problem is formulated as:

$$\begin{aligned} &\text{Minimize } TC \\ &\text{subject to } TFT \leq D \end{aligned} \quad (8)$$

4 COST-EFFECTIVE DEADLINE-CONSTRAINED STRATEGY

Because the task scheduling problem in vehicle cloud is NP-hard, it is very difficult to achieve better scheduling results directly. Service cost is related to task computation time and unit price of vehicle, and the unit price of vehicle is known.

Cost-effective and deadline-constrained strategy (CEDCS) is based on the task model and the vehicle cloud system model, and the process of solving the problem was divided into two steps: first, the task computation and finish time are calculated, and the initial allocation strategy is implemented to shorten the application finish time to meet the deadline; Then the rescheduling strategy is optimized to reduce the service cost by calculating the service cost through the task computation time.

The proposed algorithm uses the following notations:

Table 1: Notation definition

Notation	Definition
T	Task set, $T=\{t_1,t_2,...,t_k\}$
V	Vehicle node set, $V=\{v_1,v_2,...,v_m\}$
l_i	Level whose serial number is i
Level(t_i)	Level id of t_i
I	Task dependence matrix
State	Status value of a node
tasksInLevels	Tasks within each level
SD	Level-based task subdeadline
P	Task scheduling result

4.1 Initial Allocation Solution

Firstly, the scheduling method based on max-min algorithm and level partition is used. The DAG is divided into multiple levels, which simplifies the complex dependency between tasks and improves the parallelism of tasks. The shortest path algorithm is used to calculate the transmission hop of two vehicle nodes, and then the transmission hop is mapped to the transmission rate. A vehicle is considered available if it is valid within the vehicle cloud range and multiple tasks do not compete for the same node. The maximum and minimum task finish time of each level is calculated and a feasible task scheduling scheme is obtained. In order to meet the deadline, the finish time needs to be further shortened. By dividing the deadline into sub-deadlines according to the level, the task is constantly adjusted until the deadline is met.

The proposed algorithm is as follows:

ALGORITHM 1: Initial Allocation Solution

Input: T, V, I, D, tasksInLevels
Output: P
1: The number of levels is levelNum
2: **for**(i=1; i<=levelNum; i++)
3: **for**(j=1; j<=taskNum; j++)
4: if(first allocation)
5: **for**(k=1; k<=M; k++)
6: Calculate $T_p(t_j)$, $T_c(t_j)$, EST(t_j) and FT(t_j);
7: Calculate State(v_k), get the minimum allocation;
8: **endfor**
9: Get the maximum allocation;
10: Update P and remain tasks;
11: else
12: **for**(k=1; k<=M; k++)

13: Calculate AST of queuing tasks, and update FT;
14: Update State, get the minimum allocation;
15: **endfor**
16: Get the maximum allocation;
17: Update P and remain tasks;
18: **end for**
19:**end for**
20:if(TFT>D)
21: Adjust tasks in sequence. If the adjustment is successful, update P;
22: If the application finish time is less than D, return P;
23:return P;

The bottom-up Approach (BUA) [15] is adopted when assigning the level of the task. Level whose serial number is i is denoted as l_i, and Level id of t_i is denoted as Level(t_i). The calculation method of level based on the precursor dependency of the task is formulated as:

$$\text{Level}(t_i)=\begin{cases} 1 & ,\text{if } t_i = t_{entry} \\ \max_{t_p \in \text{Pred}(t_i)}\{\text{Level}(t_p)\}+1 & ,\text{otherwise} \end{cases} \quad (9)$$

According to the order of level, max-min algorithm [16] will calculate the State for each task assigned to a vehicle node, and obtain the mapping from the task to the node according to the State. State(v_k) is formulated as:

$$\text{State}(v_k)=\begin{cases} \text{FT}(t_j) & ,\text{if } v_k \text{ is available} \\ \text{Unavailable} & ,\text{otherwise} \end{cases} \quad (10)$$

The difference between first assignment and other assignments is that: Considering that the task is assigned to the vehicle node of the processed task, it needs to queue up on this node and wait for the completion of other tasks before starting to process. At this point, the actual start time (AST) of queuing task is equal to the finish time of the previous task. AST(t_i) is formulated as:

$$\text{AST}(t_i)=\text{FT}(t_j) \quad (11)$$

At this time, FT(t_i) is formulated as:

$$\text{FT}(t_i) =\text{AST}(t_i)+T_p(t_i) \quad (12)$$

According to the calculated State, a task scheduling scheme is obtained based on max-min algorithm. In order to meet the deadline requirement, tasks need to be adjusted. Adjust one task at a time, keep the assignment of other tasks unchanged, calculate the estimated finish time (EFT) of the task under this adjustment scheme, and calculate the estimated total finish time (ETFT). The sub-deadline of l_i is SD(l_i) [17]. There are two necessary conditions for successful task adjustment. The first condition is that after adjustment EFT()<SD(), or EFT()>=SD() but ETFT<TFT. The other is that all assigned vehicles are available after vehicle availability test. The algorithm continually adjusts tasks until the deadline is met or all tasks are traversed.

4.2 Task Rescheduling Algorithm

We propose a rescheduling algorithm for low service cost, which takes deadline as constraint and minimizes service cost as optimization objective. The algorithm considers the queuing before and after the intra-level rescheduling and the

queuing phenomenon caused by cross-level tasks. The processing time and service cost of tasks are updated according to the order of level and the application finish time is calculated. If the service cost is reduced and the application finish time still meets the deadline, the rescheduling algorithm succeeds.

When a task is rescheduled, a task is assigned to two different vehicle nodes under rescheduling, which will cause the time change of queuing tasks on two nodes. The queuing task sequence on v_j in l_i is denoted as $Seq(l_i, v_j)$, and the execution sequence of queuing tasks in the original allocation scheme can be determined according to $Seq(l_i, v_j)$. The queuing situation of the two nodes is different. If multiple tasks need to be queued on a new node, the node processes the tasks with the earliest start time according to the FCFS principle. In this way, the node can be processed in time when the task needs to be processed first, and the resource utilization rate of the node can be improved. When there are multiple tasks queuing on the original node, the queuing order of these assigned tasks is kept unchanged and the time is recalculated in this order.

For subsequent levels of rescheduling, the earliest start time of the next level is related to the task finish time of the previous level. The earliest start time of all tasks in the next level must be updated first. In addition, because rescheduling will change the nodes to which tasks are assigned, cross-level tasks may queue up on the same node. Since this rescheduling scheme traverses tasks in the order of levels, tasks at subsequent levels should queue up after tasks at previous levels. According to this rule, the task start and finish times are updated, and vehicle availability is checked, with other task assignments unchanged.

The proposed algorithm is as follows:

Algorithm 2: Rescheduling Algorithm

```
Input: T, V, I, D, tasksInLevels, original P
Output: updated P
The number of levels is levelNum
for(i=1; i<=levelNum; i++)
  for(j=1; j<=taskNum; j++)
    Get t_m with the largest load and reschedule it;
  endfor
  for(j=1; j<=M; j++)
    Calculate T_p(t_m), T_c(t_m), EST(t_m) and EFT(t_m);
    Calculate ETFT and ETC;
    if(ETC<TC&&ETFT<=D)
      This rescheduling scheme is feasible, and this rescheduling scheme will be retained and added into r;
  endfor
  if(r isn't empty)
    Find the node v in r that minimizes ETC;
    The rescheduling scheme t_m->v succeeds, update P;
  else
    Rescheduling fails;
  Mark t_m "processed" and add into q, then iterate through the next task in p;
endfor
return P;
```

In Algorithm 2, p is an unmarked task queue within level, and q is a processed task queue within level, which is used to distinguish whether tasks have been scheduled. r is a feasible node queue in a rescheduling. When multiple nodes meet the requirements, the scheduling scheme that minimizes ETC is selected. For all tasks in DAG, the rescheduling algorithm for low service cost adopts iterative rescheduling in order of level. For the same level, tasks are rescheduled in descending order of load size. Heavy load tasks are rescheduled first. The goal is to achieve a lower total service cost and reduce the computation complexity of rescheduling other tasks.

5 PERFORMANCE EVALUATION

In order to verify the effectiveness of CEDCS, this chapter will conduct simulation experiments from two aspects of service success ratio and service cost by comparing with the existing genetic algorithm (GA). In order to ensure the accuracy and scientific nature of the experimental results, the data in the experiment are taken as the average value of 100 experiments.

5.1 Simulation Settings

The simulation experiment simulates a scene of a straight road section in expressway, where vehicles move at almost same speed. Vehicles start from different locations on this road section and drive one-way from left to right, while keeping the direction and speed of vehicle unchanged. Due to different initial location, vehicles have different available time intervals. It can be seen that all inter-vehicle transmission rates can be calculated before scheduling. Like the processing capacity of vehicles, they are static values, which conform to the research direction of static scheduling in this paper. In this experiment, there is only one RSU on the road section. Once a vehicle leaves the RSU range, it will not be able to complete the transmission and computation of user's tasks.

The requested application is represented by DAG. RSU performs the proposed task scheduling strategy. Application finish time and total service cost are two important results obtained in this experiment. By comparing application finish time and deadline, we can judge whether the service is successful. By changing the initial position of the vehicle node, that is, changing the network topology of the vehicle node, and other experimental conditions remain unchanged, the service success ratio of multiple services can be calculated for multiple requests from user to the vehicle cloud.

DAG in the simulation experiment is shown in Figure 1. Simulation experiment uses following load of DAG (the unit of computation load is GCycles and the unit of transmission load is Mb)

Table 2: Load of Application

Task	L_p	L_c	Task	L_p	L_c
t_1	3	2.5	t_{12}	3	2
t_2	2	1.5	t_{13}	4	2
t_3	3	2.4	t_{14}	3	2.7
t_4	4	3	t_{15}	2	1.5
t_5	3	2	t_{16}	1	0.8
t_6	5	3	t_{17}	5	4
t_7	1	0.9	t_{18}	2	1.5
t_8	4	3.5	t_{19}	3	2.5
t_9	2	1.5	t_{20}	3	2
t_{10}	5	3	t_{21}	4	3.6
t_{11}	3	2			

Wherein, the unit of computation capacity is GHz, the unit of transmission rate is Mbps, and the calculation method of unit price of vehicle is UP(v_i)=Cap(v_i) ^2*1.2+0.8. For the aforementioned application, deadline is set to 5s for the first test and increased correspondingly for the following test, while keeping the transmission load unchanged. The information of vehicular nodes is shown in figure 3.

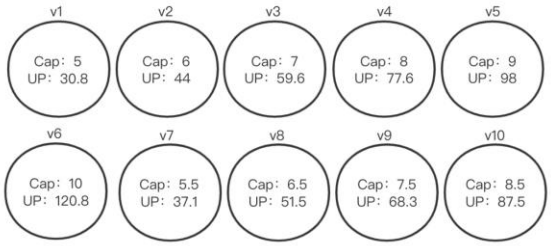

Figure 3: Information of vehicular nodes

5.2 Simulation Results

This section visually compares the performance differences between CEDCS and GA in this paper by using two performance indicators, namely service success ratio and service cost, in the form of bar chart and broken line chart. Each experiment changed the initial position of the vehicle node, resulting in the change of the transmission rate between vehicles and the effective time of vehicles in vehicle cloud. The service success ratio and average service cost of the two algorithms in the 100 experiments were calculated respectively. Where, the number of individuals in GA population is 20, and the number of genetic iterations is 15.

5.2.1 Service Success ratio

The service success ratio is shown in Figure 4, and the horizontal coordinate is the value of computation load. Since computation load increases form 65 to 125 (increased by 15 each time). According to the experimental result, CEDCS has a higher service success ratio than GA under the same computation load. In addition, with the increase of computation load, the service success ratio of CEDCS can be maintained in a certain range, while GA has a large fluctuation. This is due to the randomness of GA algorithm itself, resulting in unstable scheduling results.

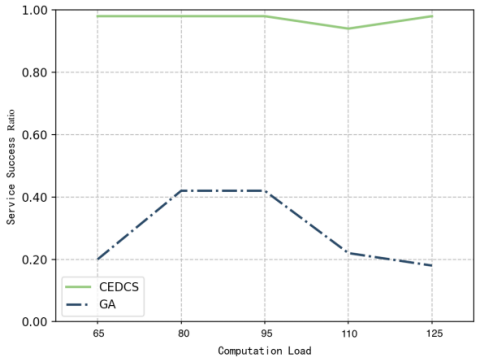

Figure 4: Service Success Ratio

5.2.2 Service Cost

The service cost is shown in Figure 5. According to the experimental result, CEDCS has a lower service cost than GA. Moreover, with the increase of computation load, the service cost of CEDCS is lower than GA. When calculating the average service cost, the experimental data is selected as the scheduling result that can meet the deadline first, so as to meet the goal of this paper.

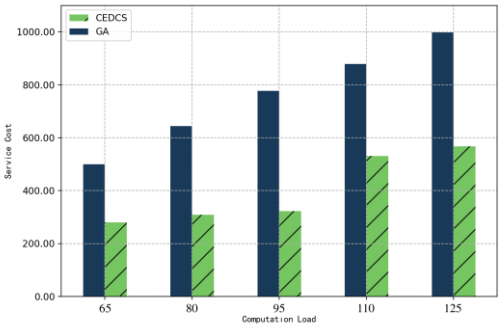

Figure 5: Service Cost

6 CONCLUSION

In this paper, we study the characteristics of vehicular cloud and utilize vehicular cloud to provide service for user. We propose a cost-effective and deadline-constrained strategy (CEDCS) to meets the deadline and get lower service cost. The task model and vehicle cloud system model are established, and we considers two processes of task transmission and computation in the vehicle cloud. Aiming at the task scheduling problem, an initial allocation strategy for completion time constraint is proposed. In the initial allocation strategy, a scheduling method based on max-min algorithm and level partition is firstly used to complete the calculation of the maximum and minimum completion time of tasks within each level by dividing DAG into levels, calculating the transmission rate between vehicles and checking the availability of vehicles. Then we use a scheduling method that shortens the completion time to meet the user-defined deadline. In order to reduce the service cost, a rescheduling algorithm for low service cost is proposed based on the initial allocation strategy. CEDCS is compared with GA by simulation experiments to verify that CEDCS has higher service success ratio and lower service cost.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (grant No. 61802181).

REFERENCES

[1] Vaquero L M, Rodero-Merino L, Caceres J, et al. A break in the clouds: towards a cloud definition[J]. ACM sigcomm computer communication review, 2008, 39(1): 50-55.

[2] Fernando N, Loke S W, Rahayu W. Mobile cloud computing: A survey[J]. Future generation computer systems, 2013, 29(1): 84-106.

[3] Hashemi S M, Bardsiri A K. Cloud computing vs. grid computing[J]. ARPN journal of systems and software, 2012, 2(5): 188-194.

[4] Foster I, Zhao Y, Raicu I, et al. Cloud computing and grid computing 360-degree compared[C]//2008 grid computing environments workshop. Ieee, 2008: 1-10.

[5] Gawali M B, Shinde S K. Standard deviation based modified cuckoo optimization algorithm for task scheduling to efficient resource allocation in cloud computing[J]. J. Adv. Inf. Technol, 2017.

[6] Bansal S, Hota C. Efficient refinery scheduling heuristic in heterogeneous computing systems[J]. Journal of Advances in Information Technology,

Academy Publisher, August, 2011, 2(3): 159-164.

[7] Topcuoglu H, Hariri S, Wu M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE transactions on parallel and distributed systems, 2002, 13(3): 260-274.

[8] Bao L, Wu C, Bu X, et al. Performance modeling and workflow scheduling of microservice-based applications in clouds[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(9): 2114-2129.

[9] Faragardi H R, Sedghpour M R S, Fazliahmadi S, et al. GRP-HEFT: A budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(6): 1239-1254.

[10] Meena J, Kumar M, Vardhan M. Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint[J]. IEEE Access, 2016, 4: 5065-5082.

[11] Sun F, Hou F, Cheng N, et al. Cooperative task scheduling for computation offloading in vehicular cloud[J]. IEEE Transactions on Vehicular Technology, 2018, 67(11): 11049-11061.

[12] Florin R, Olariu S. Toward approximating job completion time in vehicular clouds[J]. IEEE Transactions on Intelligent Transportation Systems, 2018, 20(8): 3168-3177.

[13] Sahni J, Vidyarthi D P. A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment[J]. IEEE Transactions on Cloud Computing, 2015, 6(1): 2-18.

[14] Arabnejad V, Bubendorfer K, Ng B. Budget and deadline aware e-science workflow scheduling in clouds[J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 30(1): 29-44.

[15] Han P, Du C, Chen J, et al. Minimizing Monetary Costs for Deadline Constrained Workflows in Cloud Environments[J]. IEEE Access, 2020, 8: 25060-25074.

[16] Etminani K, Naghibzadeh M. A min-min max-min selective algorihtm for grid task scheduling[C]//2007 3rd IEEE/IFIP International Conference in Central Asia on Internet. IEEE, 2007: 1-7.

[17] Sun T, Xiao C, Xu X. A scheduling algorithm using sub-deadline for workflow applications under budget and deadline constrained[J]. Cluster Computing, 2019, 22(3): 5987-5996.