A Novel Cost-Aware Load Balancing Algorithm for Road Side Units in Internet of Vehicles

Shivank Thapa*, Swagat Ranjan Sahoo*, Moumita Patra*, Arobinda Gupta†
*Dept. of Computer Sc. and Engg., Indian Institute of Technology Guwahati, Assam, India
{sthapa, swaga176101011, moumita.patra}@iitg.ac.in
†Dept. of Computer Sc. and Engg., Indian Institute of Technology Kharagpur, West Bengal, India
{agupta}@cse.iitkgp.ac.in

***Abstract*—Vehicular ad-hoc networks formed in an Internet of Vehicles scenario can enable many useful applications and services. Many of these applications may generate a large amount of data, which needs to be processed within some deadline to be useful. Limited resources present in vehicles may not be sufficient for processing such data. The resources present in Road Side Units (RSUs) can be used for this purpose by running Virtual Machines (VMs) there on behalf of the vehicles. However, RSUs can also become overloaded in a dense vehicular scenario if all vehicles use the services of their nearby RSUs only. Also, use of RSUs may incur a cost. Hence the combined total resources of the RSUs need to be carefully managed to ensure that a large number of VMs complete within their deadline while incurring a lower cost. In this paper, we propose an algorithm called Cost Aware Load Balancing (CALB) algorithm that assigns and executes VMs in different RSUs in the total RSU pool. The proposed algorithm aims to maximize the number of VMs that complete execution within their deadline and also attempts to minimize the overall cost incurred by VMs for using RSUs' resources. Performance of CALB is compared with several existing algorithms to show that it works better than the existing algorithms with respect to several performance metrics.**

***Index Terms*—VANET, Road side units, Load balancing, VM migration, rent-out cost.**

I. Introduction

Vehicular Ad-hoc NETwork (VANET) has emerged as a major area of research in recent times. In vanets, On-Board Units (OBU) in vehicles can communicate wirelessly with Road-Side Units (RSU) placed at various locations along the roads. VANET has enabled the prospect of building applications for providing many different types of useful services for on-road vehicles, including road safety, navigation, traffic monitoring, entertainment, etc. [1]. Some of these applications generate a large amount of data that needs to be processed in real-time as the data is very likely to hold significance for a short period of time. In many cases, OBUs in vehicles may not be able to process or even store such a large amount of data and require some external resources to store and complete the required processing of the data within time [2]. In such cases, data generated by applications can be offloaded to nearby RSUs on its path, which usually can be equipped with both a larger amount and better quality of resources as compared to those in vehicles. The RSUs can then run Virtual Machines (VMs) on behalf of the vehicles to process their data. However, an RSU is still going to have limited resources and can become overloaded. For example, RSUs in a dense traffic region or near intersections may receive requests from a large number of vehicles that are beyond their serving capacity, while some other RSUs elsewhere are less utilized. Due to this, some requests may get dropped and/or fail to complete, resulting in a deteriorated quality of service. One way to address this problem would be to efficiently distribute the load from overloaded RSUs to the less utilized RSUs by migrating the VMs among RSUs [3] [4]. However, VM migration has a cost associated with it and can also result in increasing the delay in serving application requests [5] [6]. In addition to the cost of migration, using the resources of RSUs can also incur cost for usage, with different RSUs incurring possibly different costs. Hence, the problem of increasing the number of VMs that complete their execution within time while reducing the total cost is challenging.

In this work, we address the problem of managing the services of RSUs to maximize the number of VMs completing their execution while minimizing the total cost, i.e., the sum of rent-out cost and migration cost. Rent-out cost is the cost associated with using RSU's resources by the VMs and migration cost is the cost of migrating VMs among the RSUs. We first formally specify the problem and propose a pricing model for modeling the rent-out cost of RSUs. We then formulate the problem as an assignment problem on a weighted bipartite graph, and propose an algorithm for it called Cost Aware Load Balancing (CALB) that assigns VMs to RSUs while considering the storage and computation capacity, DPR, and rent-out cost of RSUs. The performance of CALB is compared both with two other existing algorithms and a lower bound (for cost) scenario with respect to VM completion percentage, delay, rent-out cost, and total cost. It is seen that CALB outperforms existing algorithms and performs close to the lower bound scenario for both periodic applications that generate data periodically and event-driven applications that generate data on specific events only.

II. Related Work

Applications with deadlines can be classified broadly into two classes - delay-sensitive and delay-tolerant. Migrating delay-tolerant applications to RSUs with less load to balance the load in vehicular environments is used in [1]. The works in [6] and [7] show that prior knowledge of RSU's and vehicles' positions can improve the load balancing performance. System performance can also be improved by partitioning the tasks

978-3-903176-51-5 © 2022 IFIP

and executing a part of the task in the vehicle and the remaining part in the RSUs [2]. Algorithms for load balancing using VM migration based on the available storage and computation are given in [8] and [9]. Renting out resources is a useful way to minimize the setup cost of server infrastructure in cloud federations already and related works in this area have some similarities with the problem we address [8], [10]–[12]. In vehicular networks, VMs of fog nodes are rented to increase the reliability of the system as mentioned in [13]. In [8] and [9], load balancing algorithms are proposed by considering only storage and computing resources, but in the current work, we consider rent-out cost and data processing rate of RSUs in addition to storage and computing resources of RSUs. In [12], a dynamic resource pricing model is proposed for sharing resources between the cloud providers. To minimize the cost of the vehicular service provider a cost and delay-aware resource allocation algorithm is proposed. In the current work, the focus is on the resources of RSUs and the cost associated in using the resources. To the best of our knowledge, no work in the literature has tried to maximize the number of tasks completed and minimize the cost at the same time.

III. Problem Formulation

We consider a city scenario with a set of vehicles, where each vehicle runs one application at a time that generates data at different time instants. We assume that the amount of data generated by the applications running in these vehicles is known. Applications have specific deadlines. Data generated by these applications should be processed within the given fixed deadline. There is a set of RSUs with non-overlapping coverage areas which are interconnected via some backbone networks. These RSUs have fixed storage and computing resources that are larger than the amount of storage and computing resources available in any vehicle's OBU. The data generated by the vehicles are temporarily stored in the vehicles and are then transferred to the RSUs for processing when a vehicle comes in contact with the RSUs in its path. The resources available in the RSUs are rented out to vehicles to run VMs for processing the data generated by the applications in the vehicles. A VM is said to have completed its execution if all data generated by the corresponding application is processed within some deadline.

Let the total time under consideration be T, with the time slots numbered from 0 to $T-1$. We denote the set of vehicles as $V = \{v_1, v_2, \ldots, v_X\}$ and the set of RSUs as $R = \{r_1, r_2, \ldots, r_Y\}$. Each vehicle $v_j \in V$ can be represented by a tuple $\langle P_j, a_j, \mathcal{L}_j, \wedge_j \rangle$, where P_j is the path followed by v_j represented as a sequence of RSUs $\langle r_1^j, r_2^j, \ldots, r_K^j \rangle$ encountered by the vehicle on the path, a_j is the application being run by v_j, $\mathcal{L}_j$ is the lifetime of the application running at v_j, and $\wedge_j$ is the data generation rate of application running at v_j. The data generation rate $\wedge_j$ is represented by a sequence $\langle \lambda_1^j, \lambda_2^j, \ldots \lambda_T^j \rangle$, where $\lambda_k^j \geqslant 0$ is the amount of data generated by vehicle j at time slot k. Further, for each RSU r_k^j in the path of v_j, its arrival and departure time is denoted by $arr_{r_k}^j$ and $dep_{r_k}^j$ respectively. We represent an RSU r_i by the tuple $\langle p_i, s_i, D_i^r \rangle$, where p_i is the computing capacity, s_i is the storage capacity and D_i^r is the data processing rate of r_i. Finally we have the set of applications, denoted as $A = \{a_1, a_2, \ldots, a_k\}$. Each application a_k can be represented by a tuple $\langle p_k^j, s_k^j, Cost^f \rangle$, where p_k^j is the amount of computing need for the application a_k in vehicle v_j, s_k^j is the amount of storage needed for the application a_k in vehicle v_j, and $Cost^f$ is the fixed cost of migration. The deadline for an application varies based on the vehicle it runs in, and is defined as the time the vehicle leaves the last RSU in its path. The output is defined by two sets of variables $\{x_{ijt}\}$ and $\{y_{ijt}\}$, $\forall r_i \in R$, $\forall v_j \in V$, and $\forall t, 1 \leqslant t \leqslant T$. For a VM of vehicle v_j, the variable x_{ijt} is set to 1 if it is present in RSU r_i at time t, otherwise it is 0. Variable y_{ijt} is set to 1 if it is scheduled in RSU r_i at time t, otherwise it is 0. The amount of unprocessed data for the VM of vehicle v_j at time t can be computed as the difference between the data transferred by the vehicle till time t and the data processed by RSUs till time t for vehicle v_j. This can be written as,

$$U_t^j = \sum_{k=1}^{arr_{r_c}^j} \lambda_k^j - D_i^r \sum_{i=1}^{Y} \sum_{t=1}^{T} y_{ijt} \tag{1}$$

The variable $arr_{r_c}^j$ is the arrival time of vehicle v_j at the last RSU in the vehicle's path before time t. We define an indicator variable $z_j^{t,k,i}$, whose value is set to 1 if and only if $(x_{kjt} = 1 \ \wedge \ x_{ijt_1} = 1 \ \wedge \ t_1 = t+1 \ \wedge \ k \neq i)$, 0 otherwise. Thus $z_j^{t,k,i}$ indicates that VM j has migrated from RSU k to any other RSU i at time t. Therefore, $z_j^{t,k,i} = 1$ if at time t, VM j migrated from RSU k to i, 0 otherwise.

If $k = i$, the VM is not migrated to any other RSU and the migration cost incurred will be zero. We also define the indicator variable I_j, such that, $I_j = 1$ if $U_t^j = 0$ when v_j leaves the last RSU in its path, 0 otherwise.

We can now formally define the problem as,

$$maximize \sum_{j=1}^{X} I_j \tag{2}$$

$$minimize \sum_{j=1}^{X} \sum_{t=1}^{T} (z_j^{t,k,i})(Q_i^t + M_i^t) \ + \ (1 - z_j^{t,k,i})(Q_k^t) \tag{3}$$

subject to the following constraints:

$$\forall v_j \in V, \forall t > arr_{r_l}^j, \quad \lambda_t^j = 0 \tag{4}$$

$$\forall v_j \in V, \forall t, 1 \leqslant t < arr_{r_1}^j, \quad x_{ijt} = 0 \tag{5}$$

$$\forall v_j \in V, \forall t, arr_{r_1}^j \leqslant t \leqslant dep_{r_l}^j, \quad \sum_{j=1}^{Y} x_{ijt} = 1 \tag{6}$$

$$y_{ijt} = 1 \Longrightarrow (x_{ijt} = 1 \wedge U_t^j \neq 0) \tag{7}$$

$$\forall r_i \in R, \forall v_j \in V, \ \ \forall t, 1 \leqslant t \leqslant T, \ \ \sum_{j=1}^{X} p_k^j \, y_{ijt} \leqslant p_i^r \tag{8}$$

$$\forall r_i \in R, \forall v_j \in V, \forall t, 1 \leqslant t \leqslant T, (\sum_{j=1}^{X} U_t^j + \sum_{j=1}^{X} s_k^j) x_{ijt} \leqslant s_i^r \tag{9}$$

$$\forall v_j \in V, (\sum_{t=1}^{arr_{r_l}^j} \lambda_t^j)/(D_i^r \sum_{i=1}^{Y} \sum_{t=1}^{dep_{r_l}^j} y_{ijt}) \leqslant dep_{r_l}^j \tag{10}$$

Equation 2 and Equation 3 represent the objective functions to maximize the number of VMs completing execution and to

minimize the total cost incurred in placing, scheduling, and migrating the VMs in the system respectively. The constraint in Equation 4 limits the data generated by the vehicles when they reach the last RSU (r_l^j) in their route i.e. data generated after this point is not considered for processing. Equation 5 indicates that no VM will be created for a vehicle until its arrival time at the first RSU in its route. Equation 6 makes sure that the VM must be present in exactly one RSU at any time t, between the arrival to the first RSU and departure from the last RSU in its path. Equation 7 ensures that a VM is scheduled only when it has some unprocessed data. Equation 8 and Equation 9 specify that the total computing and storage need of all the VMs combined at an RSU should not exceed the total computing and storage capacity of the RSU respectively. Equation 10 restricts the total execution time required by a vehicle's VM to be less than the departure time of vehicle from last RSU (r_l) in its path.

The problem formulated is a mixed-integer nonlinear programming problem, which is NP-hard. Therefore, we next propose a heuristic algorithm, Cost Aware Load Balancing (CALB), for the problem.

IV. Cost Aware load Balancing (CALB)

In this Section, we first introduce a fixed pricing model for rent-out cost of the RSUs. We then propose a graph theoretic formulation for solving the problem and finally specify our proposed algorithm called Cost Aware Load Balancing (CALB). CALB focuses on maximizing the number of VMs completed while minimizing the overall cost.

A. Pricing Model

We consider a fixed pricing model for RSUs to rent out resources, where each RSU has a predefined cost associated with its computing and storage resources.

For every RSU $i \in R$, we assume a predefined cost associated with its per unit storage resources per unit time (S_i) and per unit computing resources per unit time (C_i). We also assume a predefined per unit time base cost (P_i) when a VM is present at an RSU waiting for the data of the corresponding application to arrive. Using the above definitions, we define Q_i^t as the rent-out cost offered by RSU i at time step t, based on a VM's storage and computing requirements. The following cases are possible for the cost incurred to process a VM:

Case 1: *When a VM has no data to process but has not completed its execution, i.e., the VM is waiting for more data from the application to process.* The rent-out cost for a VM j waiting for some data in RSU i at time t is the base cost.

$$Q_i^t = P_i \tag{11}$$

Case 2: *When a VM j, present in RSU i at current time step t has some unprocessed data but is not scheduled to execute at this time step.* The rent-out cost for a VM j is given as:

$$Q_i^t = U_t^j S_i \tag{12}$$

where, U_t^j is the unprocessed data in VM j at time t and S_i is the cost per unit storage resources per unit time.

Case 3: *When a VM j present in RSU i at the current time step t has some unprocessed data and is also scheduled to execute at time step t.* The computing cost in addition to storage cost for the VM's unprocessed data is given as:

$$Q_i^t = U_t^j S_i + V^j C_i \tag{13}$$

where, V^j is the computing requirement of the VM j.

We also consider the cost incurred for migration of VM j to RSU r_i at time t (if a migration occurs at t) as:

$$M_j^t = Cost_j^f + U_t^j * Cost_j^v \tag{14}$$

where, $Cost_j^f$ is the fixed cost of migrating the VM for vehicle v_j and $Cost_j^v$ is the migration cost associated with per-unit of data transfer. The total cost incurred by a VM at time t is the sum of the rent-out cost and migration cost of the VM.

B. Graph Theoretic Formulation

The problem of assignment of VMs to RSUs at time step t is represented by a weighted bipartite graph. We add a node for every VM that has not yet completed its execution at t, and also a node for every RSU. Edges are then added from a VM to an RSU if the VM can be assigned to that RSU. Thus the set of nodes for the VMs and the set of nodes for the RSUs form two partitions in the bipartite graph. Dummy nodes are added to either set as needed to make the number of nodes the same in both partitions, as our algorithm will need this property. This graph is shown in Figure 1, with X VMs and Y RSUs. The goal is to assign each VM to one RSU in the RSU set at t while trying to maximize the number of VMs completing and minimize the total cost. To this end, we assign a weight to each edge as follows:

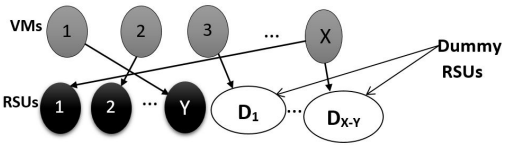

Fig. 1: Graphical representation

$$\begin{aligned} W_{ji} = \omega_1(R_t[i].s - (D_j^g + U_t^j)) \ + \omega_2(D_i^r) \ + \\ \omega_3(-R_t[i].p/V[j].p) \ + \ \omega_4(-R_t[i].Q_i^t) \end{aligned} \tag{15}$$

Here, $0 \leqslant \omega_1, \omega_2, \omega_3, \omega_4 \leqslant 1$ and $\omega_1 + \omega_2 + \omega_3 + \omega_4 = 1$. These $\omega_1, \omega_2, \omega_3, \omega_4$ are the weights that assign priorities to storage resources, data processing rate, computing resources, and rent-out cost, respectively. W_{ji} is the weight of the edge between VM node j and RSU node i at time t. $R_t[i].s$ is the amount of storage available with RSU i. D_j^g is the expected data generated by application j. D_i^r is the data processing rate of RSU i. U_t^j is the amount of unprocessed data in VM j. $R_t[i].p$ is the amount of computing resources available with RSU i. $V[j].p$ denotes the computing need of application of VM j. $R_t[i].Q_j^t$ is the maximum rent-out cost offered by RSU i to VM j. The different terms in Equation 15 are described as follows:

- The term $(R_t[i].s - (D_j^g + U_t^j))$ indicates the difference between storage available with RSU i and amount of unprocessed data in VM j. The expected data that can be generated by the application till the deadline is added

to the unprocessed data at current time step to find a suitable RSU. The generated data can be calculated as $D_j^g = \sum_{k=t}^{T} \lambda_k$.

- Since the Data Processing Rate (DPR) of RSUs can vary, each VM attempts to find an RSU having a greater DPR. This is represented using the term (D_i^r).
- The term $(-R_t[i].p/V[j].p)$ represents the ratio between available computing resources of RSU and required computing need of VM. Since the computing resources required by a VM is constant, we use best-fit policy to efficiently allocate these resources.
- The term $(-R_t[i].Q_i^t)$ indicates the maximum rent-out cost offered by RSU i. To minimize the rent-out cost, the cost of the chosen RSU should be low. Therefore, we assign a negative value to the cost factor so that higher cost of an RSU will make the weight of the edge lower.

C. Proposed Algorithm

In this Section, we briefly describe the proposed algorithm, Cost Aware Load Balancing (CALB). The pseudocode of the algorithm is shown in Algorithm 1. CALB runs from time $t = 0$ till $t = T - 1$. In each iteration, it first admits the VM requests to the system based on resources available in RSUs (Lines 3-5). VMs that cannot be admitted at time t due to lack of resources are temporarily dropped and considered again at time $t + 1$ (Line 6). It then chooses a set of VMs for assignment at time t. For the assignment, a graph is first formed as described before, and then the well-known Hungarian matching algorithm is run on the graph to find a maximum-weighted matching on the graph, and the VMs are assigned to the RSUs based on the matching found (Lines 7-11). Once assignment of VMs is over, based on the available computing resource of RSUs, VMs get scheduled in the RSUs they are assigned to in earliest deadline first manner (Lines 12-19).

Algorithm 1: Cost Aware Load Balancing (CALB)

```
1  Initialize the RSUs and VMs information
2  for (t = 0; t < T; t = t + 1) do
3      Append the temporary dropped VM's request from time
         step t − 1
4      Update VM's resources requirements
5      Admit VM's requests based on resource availability
6      If no suitable RSU found, request dropped and again
         considered in time step t + 1
7      while (all chosen VMs are assigned or no assignment is
         possible) do
8          Choose a set of VMs to be assigned
9          Form the graph with weights as per Equation 15
10         Use Hungarian matching algorithm on the graph
11     Update the assigned VMs, RSUs information
12     for (each r_i in R) do
13         Sort the VMs present in r_i based on earliest
             deadline first manner
14         for VM j in r_i's VM's list do
15             if (VM j is not complete) then
16                 if (sufficient computing resources is
                     available at r_i) then
17                     Execute VM j
18             else
19                 Add the VM j to completed VMs list
```

V. Simulation Results

The proposed algorithm CALB is simulated using a Java-based discrete event simulator to evaluate its performance. We have considered a lower Manhattan city scenario with bidirectional roads and an area of 10 square kilometers. Nine uniformly placed RSUs are considered with transmission range 500 meters. Thus, there are many areas in the scenario that are not covered by any RSUs. Vehicle movement in the scenario is generated using Simulation of Urban MObility (SUMO) [14]. The parameters used for simulation are given in Table I. We have evaluated our proposed algorithm for both periodic and event-driven applications. A periodic application is an application that generates data periodically, while an event-driven application generates data only on specific events such as accidents in the city, arrival of the current vehicle to market areas, etc.. We have compared the results of CALB with Matching-based Dynamic Load Balancing (MDLB) algorithm [8], Joint algorithm for Selection decision, Computation resource, and Offloading ratio (JSCO) [2], and one Lower Bound (LB) scenario. For LB scenario, we have considered a hypothetical RSU with infinite storage and computing resources, and a rent-out cost that is the minimum among all RSUs. Therefore, all vehicles in this scenario can immediately transfer their requests and any data generated to this RSU for processing.

TABLE I: Table of parameters [1] [5]

Parameters	Values
Number of vehicles	50-300
Total time steps	890
Data processing rate of RSU	8-15 Mbps
Per unit storage cost of RSU	0.001$ - 0.007$
Per unit computing cost of RSU	0.001$ - 0.007$
Per unit time base cost of RSU	0.001$ - 0.004$
Fixed migration cost of VM	0.01$
Per unit cost of data transfer	0.002$
Initial storage need of VMs	100 MB- 300 MB
Computing need of VMs	5 MHz - 40 MHz
Total storage capacity of each RSU	6000 MB
Total computing capacity of each RSU	1000 MHz

Figure 2 shows the performance of algorithms for periodic applications. Figure 2(a), 2(b), 2(c), and 2(d) show the variations of VM completion percentage, average delay, average rent-out cost, and average total cost respectively with an increasing number of vehicles. It can be seen that VM completion percentage is higher in the case of CALB as compared to JSCO and MDLB. This is because CALB allocates VMs to RSUs based on the expected storage needs of the VMs, thus reducing the need for unnecessary migrations. The VM completion percentage achieved by CALB is also very close to the result obtained from LB scenario. The average delay is also less in the case of CALB as compared to JSCO and MDLB. This

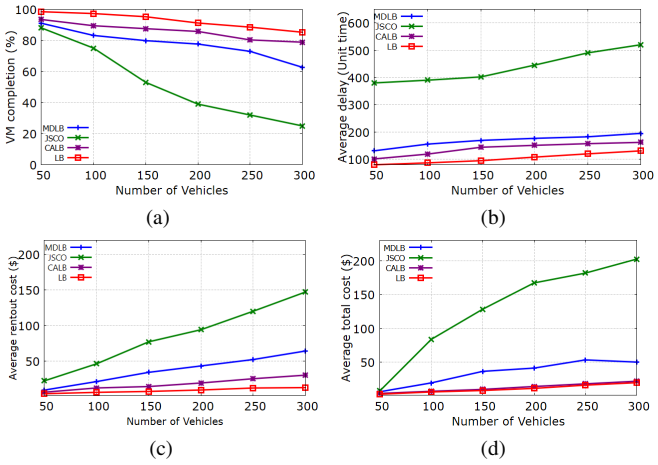

Fig. 2: Performance of algorithms for periodic applications

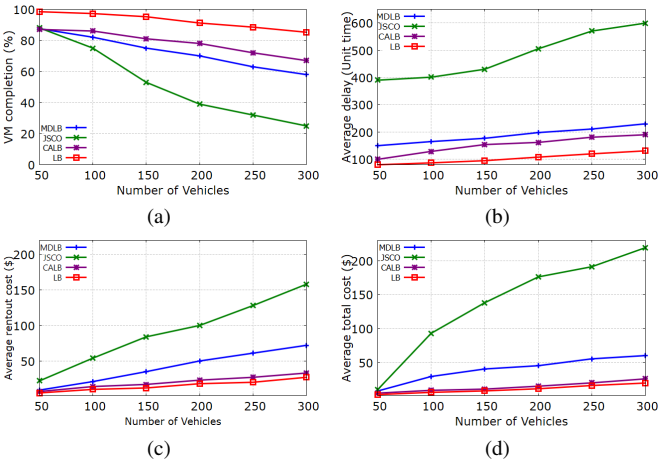

Fig. 3: Performance of algorithms for event-driven applications

is because best-fit strategy and DPR of RSUs are considered in CALB. The best-fit strategy is used for scheduling and DPR is used for assignment. The best-fit strategy minimizes queueing delay and use of RSU with higher DPR minimizes the execution time of applications. It is also observed that the average rent-out cost and the average total cost in CALB are both lower as compared to JSCO and MDLB. This is because CALB also considers pricing of the resources along with storage and computing resources, and DPR of RSUs, while MDLB only focuses on storage and computing resources and JSCO only focuses on computation. In addition, CALB also reduces the number of migrations, thus reducing the migration cost, and hence the total cost. Both the rent-out cost and total cost are also very close to the lower bound achieved in LB scenario, indicating that CALB achieves a solution that is closer to the optimal one.

We have also analyzed the performance of CALB for event-driven applications as given in Figure 3. In this case also CALB performs better than MDLB and JSCO. It is seen that VM completion percentage of event-driven applications is less in comparison to the periodic applications (Figure 3(a)). Also, the delay incurred (Figure 3(b)), the rent-out cost (Figure 3(c)), and the total cost (Figure 3(d)) are all higher in an event-driven applications in comparison to periodic applications. This is because, for event-driven applications, a large amount of data is generated during an event. With this, it becomes difficult for a VM to get a suitable RSU. This leads to an increase in the number of migrations, which eventually increases delay, decreases VM completion percentage, and increases total cost. However, CALB still performs better compared to MDLB, JSCO, and the solution obtained is closer to that found in the lower bound scenario LB.

VI. Conclusions and Future Work

In this paper, we have proposed an algorithm called CALB that aims to maximize the number of VMs completed and minimize the overall cost. Simulation results show that CALB outperforms two other existing algorithms, MDLB and JSCO, with respect to VM completion percentage, delay, rent-out cost, and total cost. Comparison with a hypothetical ideal scenario is also shown, which indicates that the solution found by CALB is closer to the ideal scenario. One possible extension of the current work can be the utilization of resources available in vehicles instead of offloading the complete task to RSUs.

References

[1] G. G. M. N. Ali, E. Chan, and W. Li, "On scheduling data access with cooperative load balancing in vehicular ad hoc networks (VANETs)," *The J. of Supercomputing*, vol. 67, no. 2, pp. 438–468, Feb. 2014.

[2] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377–4387, Jun. 2019.

[3] A. M. Maia et al., "Dynamic service placement and load distribution in edge computing," in *16th Int. Conf. on Network and Service Management (CNSM)*, Nov. 2020, pp. 1–9.

[4] T. D. Schepper et al., "Load balancing and flow management under user mobility in heterogeneous wireless networks," in *14th Int. Conf. on Network and Service Management (CNSM)*, Dec. 2018, pp. 247–253.

[5] L. Li et al., "Compound model of task arrivals and load-aware offloading for vehicular mobile edge computing networks," *IEEE Access*, vol. 7, pp. 26 631–26 640, Feb. 2019.

[6] G. G. M. Nawaz Ali et al., "An efficient cooperative load balancing approach in RSU-based Vehicular Ad Hoc Networks," in *IEEE Int. Conf. on Control System, Computing and Engineering*, Nov. 2014, pp. 52–57.

[7] M. F. Tsai et al., "Improving positioning accuracy for VANET in real city environments," *J. of Supercomputing*, vol. 71, no. 6, pp. 1975–1995, Jun. 2015.

[8] S. R. Sahoo, M. Patra, and A. Gupta, "MDLB: A matching based dynamic load balancing algorithm for road side units," in *Int. Wireless Comm. and Mobile Computing (IWCMC)*, Aug. 2021, pp. 291–296.

[9] S. R. Sahoo et al., "AALB: Application Aware Load Balancing Algorithm for Road Side Units," *Veh. Comm.*, vol. 36, p. 100475, Aug. 2022.

[10] H. Deng et al., "Revenue maximization for dynamic expansion of geo-distributed cloud data centers," *IEEE Trans. on Cloud Computing*, vol. 8, no. 3, pp. 899–913, Jul. 2020.

[11] H. Li, J. Liu, and G. Tang, "A pricing algorithm for cloud computing resources," in *Int. Conf. on Network Computing and Information Security*, vol. 1, Jul. 2011, pp. 69–73.

[12] M. Najm, M. Patra, and V. Tamarapalli, "Cost-and-delay aware dynamic resource allocation in federated vehicular clouds," *IEEE Trans. on Veh. Technology*, vol. 70, no. 6, pp. 6159–6171, Jun. 2021.

[13] J. Yao and N. Ansari, "Fog resource provisioning in reliability-aware IoT networks," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8262–8269, Jun. 2019.

[14] P. A. Lopez et al., "Microscopic traffic simulation using SUMO," in *21st Int. Conf. on Intelligent Transportation Systems*, Dec. 2018, pp. 2575–2582.