Cost-efficient Load Balancing in Cloud-assisted Vehicular Networks

Thesis submitted to the
Indian Institute of Technology Guwahati
for the award of the degree

of

Doctor of Philosophy

in

Computer Science and Engineering

Submitted by
Swagat Ranjan Sahoo

Under the guidance of
Dr. Moumita Patra

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

June 2025

Dedicated to

My Parents

and

Family members

Who always picked me up on time
and encouraged me to go on every adventure,
especially this one

Declaration

I certify that:

- The work contained in this thesis is original and has been done by myself and under the general supervision of my supervisor.
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.
- I also affirm that no part of this thesis can be considered plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises.
- I am fully aware that my thesis supervisor is not in a position to check for any possible instance of plagiarism within this submitted work.

Date: June 03, 2025
Place: Guwahati

(Swagat Ranjan Sahoo)

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor *Dr. Moumita Patra* for her consistent support, inexhaustible patience, and positive guidance during my doctoral research. I am thankful for her ethical beliefs and philosophy which made me mature not only as a scientific researcher but also as a human.

I am highly grateful to *Prof. Tamarapalli Venkatesh* for his invaluable support and encouragement throughout my Ph.D. I would also like to thank the other members of my Doctoral Committee - *Dr. John Jose* and *Prof. Ashok Singh Sairam* for their insightful comments and suggestions which made me improve the quality and clarity of my work.

I would like to thank *Prof. Arobinda Gupta* for his insightful comments and suggestions regarding my Ph.D. work. I am also thankful to the anonymous reviewers of my research work in various forums for their critical comments which helped me to add quality to my work.

I want to thank the heads of the Department of Computer Science and Engineering during my Ph.D. at IITG for allowing me to use the facilities and the available resources. I acknowledge the Technical staff of the Department of Computer Science. I would also like to thank the staff at the Academic Affairs office who were supportive in processing my applications and grant requests.

I would like to gratefully acknowledge *MHRD, Govt. of India* for the financial support rendered throughout my years of Ph.D. without which this research could not have taken shape. I would also like to acknowledge the Department of Computer Science and Engineering and the Welfare Board of IITG for the travel grants which helped me to present my research work at the national and international levels.

I would also like to thank the research scholars of the Department of Computer Science and Engineering at IITG for creating a warm atmosphere of mutual support and encouragement.

Finally, yet importantly, I would like to thank Almighty God and my family members for their unconditional love, support, caring, warmth, and profound encouragement all these years. They never doubted my intentions and wholeheartedly supported me in all my endeavours. I fall short of words to express my gratitude to them.

Certificate

This is to certify that this thesis entitled, **"Cost-efficient Load Balancing in Cloud-assisted Vehicular Networks"**, being submitted by **Swagat Ranjan Sahoo**, to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, for partial fulfillment of the award of the degree of Doctor of Philosophy, is a bonafide work carried out by him under my supervision and guidance. The thesis, in my opinion, is worthy of consideration for award of the degree of Doctor of Philosophy in accordance with the regulation of the institute. To the best of my knowledge, it has not been submitted elsewhere for the award of the degree.

Date:
Place: Guwahati

.....

Dr. Moumita Patra
Assistant Professor
Department of Computer Science and Engineering
IIT Guwahati

Abstract

Vehicular Ad hoc NETworks (VANETs) have become an important part of a smart city environment. Vehicles are equipped with on-board units which allow them to run applications and communicate with Road Side Units (RSUs). RSUs are connected to a local server with some amount of storage and computing resources to run Virtual Machines (VMs) that process the application requests generated by vehicles. They act as a cloudlet and provide cloud support to requests. These requests may have different deadlines and resource requirements like storage, computing, and content delivery. Processing the application requests at RSUs may make some of the RSUs overloaded, especially near road intersections where a larger number of vehicles are present. This significantly affects the quality of service by increasing delay and decreasing the number of requests processed. Deployment of more RSUs may reduce the chances of overloaded RSUs. However, the cost of deployment of RSUs and their maintenance cost does not allow us to add a large number of RSUs. In this scenario, it is necessary to either increase the total resource availability by using the resources from some entities in the scenario or utilize the available resources of the network efficiently. In this thesis, we propose a set of algorithms to assign the application requests to a target node in the network such that the number of requests processed is maximized while minimizing the end-to-end delay. The target node may be an RSU, Central Cloud (CC) or Parked Vehicle (PV). First, we have utilized the available resources of other RSUs by migrating the Virtual Machines (VMs) from the overloaded RSU to other RSUs with available resources. Second, we have rented the resources from other RSUs with consideration of migration cost and rent-out cost. Third, we have rented the resources from other RSUs, PVs and the CC to process the application request. In all the scenarios, we have focused on efficient management of cost such that the users and the service providers are benefited. The proposed algorithms are evaluated by extensive simulations and their performance is compared with state-of-the-art algorithms for similar scenarios.

Contents

List of Figures

List of Tables

Acronyms

AALB Application Aware Load Balancing.

AAPM Adaptive Algorithm for Profit Maximization.

CC Central Cloud.

CM Cost Model.

DCORA Distributed Computation Offloading and Resource Allocatio.

DPR Data Processing Rate.

DSRC Dedicated Short Range Communication.

ERR Efficient Resource Renting.

FPGA Field-Programmable Gate Arrays.

ITS Intelligent Transportation Systems.

JSCO Joint algorithm for Selection decision, Computation resource, and Offloading.

LB Load Balancing.

LBClient Load Balancing Client.

LBServer Load Balancing Server.

MAMTS Multiple Applications Multiple Tasks Scheduling.

MERR Modified Efficient Resource Renting.

OBU On Board Unit.

PV Parked Vehicle.

RA Request Approval.

RAA Resource Aware Assignment.

RC Remote Cloud.

RSU Road Side Units.

SC Selection Criteria.

SLA Service Level Agreement.

SP Service Provider.

SUMO Simulation in Urban Mobility.

VANETs Vehicular Ad hoc NETworks.

VC Vehicular Cloud.

VM Virtual Machine.

1 Introduction

1.1 Vehicular Ad-Hoc Networks

A Vehicular Ad-hoc NETwork (VANET) is a type of mobile ad-hoc network that enables communication between Vehicle and Vehicles (V2V), between Vehicles and roadside Infrastructure (V2I), and between Infrastructure to Infrastructure (I2I) [1] as shown in Figure 1.1. These communication modes collectively contribute to creating a connected and Intelligent Transportation System (ITS). VANETs are designed to improve road safety, traffic efficiency, and provide various services to drivers and passengers [2]. These networks rely on the wireless communication capabilities of vehicles to create a dynamic and self-organizing network without the need for a pre-existing infrastructure. It uses Dedicated Short Range Communication (DSRC) technology for communicating between the vehicles and between vehicles and infrastructure. It operates in 5.9 GHz frequency band and is specifically designed for vehicular communication. DSRC supports both point-to-point and broadcast communications. It is based on the IEEE 802.11p protocol. It is optimized for low-latency communication and high-speed data exchange in vehicular environments. Unlike traditional Wi-Fi, which focuses on high data rates over longer ranges, 802.11p prioritizes reliability and low latency for safety-critical applications.

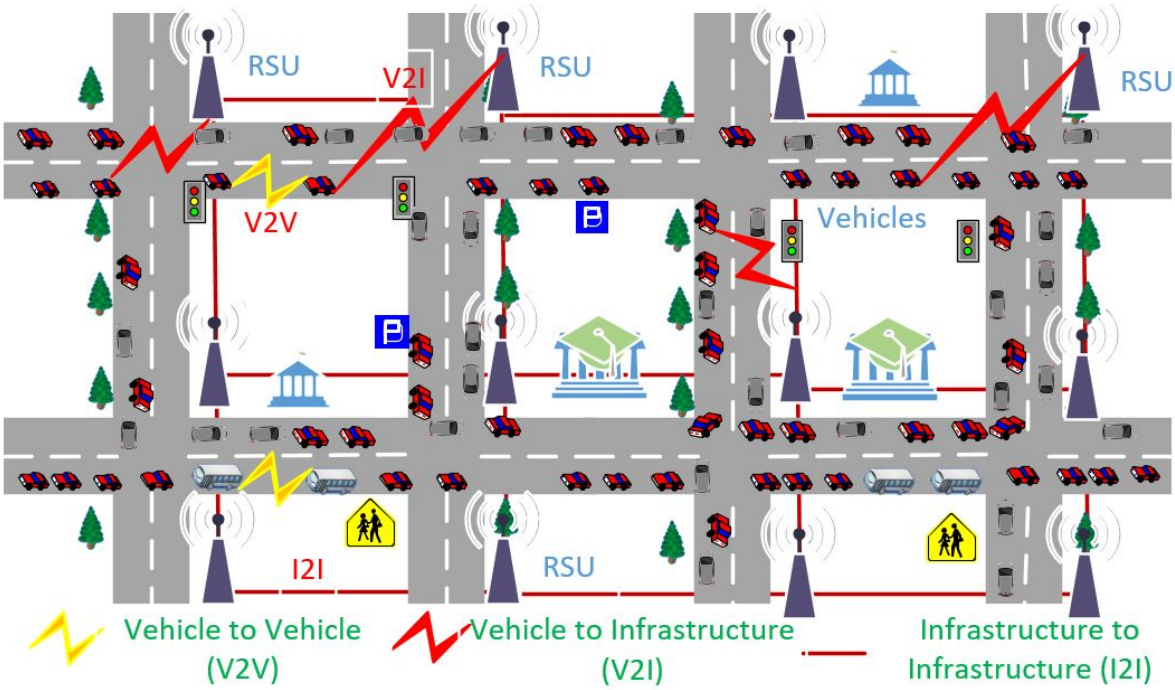


Figure 1.1: *VANET architecture*

1.1.1 Communication Modes

There are generally three communication modes in VANET. Each communication mode is discussed below:

- **Vehicle-to-Vehicle (V2V) Communication** V2V communication refers to the direct exchange of information between vehicles on the road. It enhances road safety by allowing vehicles to share real-time data about their current status, such as speed, position, acceleration, and other relevant information. Vehicles can exchange information to detect potential collision risks and take preventive actions. V2V communication enables cooperative driving, where vehicles can coordinate their movements to optimize traffic flow and reduce congestion.

- **Vehicle-to-Infrastructure (V2I) Communication** V2I communication involves the exchange of information between vehicles and roadside infrastructure elements, such as traffic lights, road signs, and other fixed installations. It improves traffic management, provides real-time information to drivers, and enhances overall road efficiency. Vehicles can receive information from traffic lights to optimize their speed

and reduce unnecessary stops, improving fuel efficiency. Infrastructure can assist by offering information about nearby services, traffic conditions, or emergencies.

- **Infrastructure-to-Infrastructure (I2I) Communication** I2I communication involves the exchange of information between different elements of the roadside infrastructure without direct involvement of vehicles. This facilitates coordination and data sharing among various infrastructure components to improve overall system efficiency. Centralized traffic management systems can collect data from various infrastructure elements to monitor and control traffic flow in real time. Coordination between different infrastructure components, such as surveillance cameras and emergency response systems, can enhance the response to possible accidents or emergencies.

Vehicles in VANETs have less capacity in terms of storage and computing resources. An increase in the number of applications in vehicles leads to the generation of huge amounts of data. The lack of availability of resources in the vehicles forces them to send the service requests to nearby Road Side Units (RSUs) which may have a higher capacity of resources. However, due to the limited amount of resources even in the RSUs, in high-traffic scenarios or road junctions, RSUs may become overloaded.

1.1.2 Load Balancing in VANETs

The growing number of applications integrated into vehicles on the roads generates vast amounts of data, surpassing the capacity for real-time storage and processing in vehicles [3]. For instance, applications such as advanced driver assistance systems, in-car entertainment, and vehicle-to-everything communication contribute to an overwhelming volume of data that exceeds the immediate capabilities for storage and real-time analysis of vehicles [4]. In this scenario, RSUs play an important role. The RSUs serve as a fixed device, similar to a compact base station, facilitating V2I communication with vehicles while simultaneously connecting to the core network through a high-speed back-haul link. Although the ideal scenario involves comprehensive RSU coverage across the entire road network, practical challenges hinder widespread deployment. One primary obstacle is the considerable investment required for making full coverage to the city environment [5]. Moreover, RSUs demand reliable electrical power sources, and safety considerations further complicate their deployment. To address these issues, researchers are exploring deployment strategies focused on achieving maximum coverage with a minimal number of RSUs. A significant drawback of static RSUs

is their inability to adapt to dynamic traffic patterns. During peak hours, RSUs may operate at maximum capacity, while in off-peak hours resources may be underutilized. This nature of RSUs becomes particularly problematic during sudden changes in road infrastructure or routine maintenance, leading to road closures and traffic diversions that impede the RSU's ability to provide consistent VANET Quality of Service (QoS). Researchers are actively addressing these challenges in the context of evolving traffic conditions and infrastructure changes. One possible solution to the overloaded scenario is through Virtual Machine (VM) migrations which helps to transfer the load from one RSU to another RSU in the network.

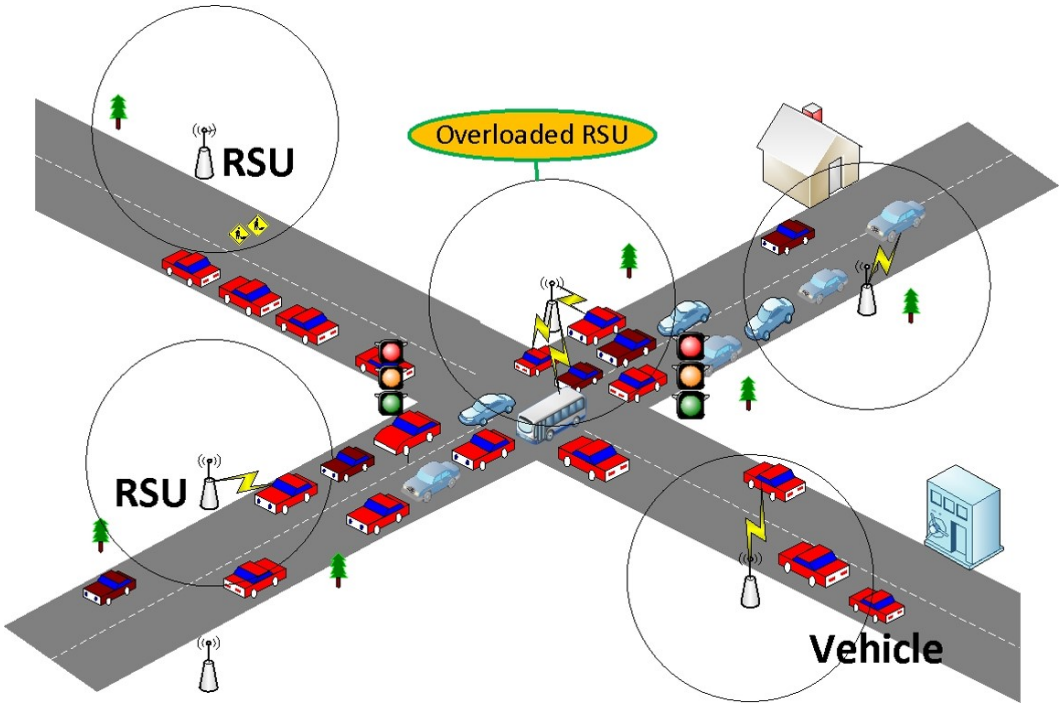


Figure 1.2: *Overloaded RSUs*

1.1.3 Virtual Machines

A VM is a compute resource that uses software to run programs and deploy applications [6]. The data generated by vehicles can be processed by running VMs in the RSUs [7]. VMs modularize task processing, enhancing security and aiding in decision making. There may be situations where a VM is migrated from one RSU to another due to an overloaded scenario, this process is known as VM migration [8]. VM migration is also costly as it

uses system resources and incurs delays, which increases the completion time of tasks. The resource capacity of RSUs becomes insufficient particularly on road junctions during peak hours which leads to an overloaded scenario as shown in Figure 1.2. Researchers have worked in this direction to use on-road vehicles, other RSUs, Parked Vehicles (PVs), and Central Cloud (CC) to avoid overloaded scenarios. However, the maximization of profit for the service provider is not focused on while doing Load Balancing (LB). In our approach, we tried to reduce the chance of overloading RSU by assigning the vehicle requests to appropriate nodes.

1.1.4 Cloud-assisted Vehicular Networks

The use of cloud resources to assist RSUs is suitable for delay-tolerant applications but in vehicular networks, various delay-sensitive applications are there which need to process the data within a very short period. Applications deployed in the remote cloud server cannot guarantee low service latency for the user because of unpredictable delays in wide area networks [9]. It affect the QoS in large extent [10]. The use of vehicles extends the capacity of the RSUs by making use of the underutilized resources of vehicles near the RSUs [11]. Use of vehicles to assist RSUs [12] [13] leads to frequent service disconnections because of the high mobility of vehicles. Notably, studies have shown that 70% of individual vehicles almost spend 95% of time in parking lots, home garages, or street parking spaces [14] [15]. PVs occupy a significant portion of the total number of vehicles present globally with rich resources. They have sufficient idle time to process offloading tasks by the RSUs [14] [16]. The idle resources combined together to alleviate the workload of the RSUs and extend the resource capacity of vehicular networks [17]. This may significantly reduce the need to add new RSUs which incurs huge infrastructure costs. However, RSUs are equipped with storage and computing resources which becomes insufficient in an overloaded scenario. The PVs and CCs can be used for processing the requests.

1.1.5 Motivation and Research Scope

An increase in the number of vehicles and applications in the vehicles generates a huge number of requests. The amount of data generated by some of the applications is huge (ex. gaming, augmented reality, image processing, etc.). Some of the applications are computationally intensive (ex. augmented reality, image processing etc.) while some are

storage-intensive (ex. video storage and transfer, storing traffic data, etc.). The deadlines for requests generated by applications are not uniform. Some are delay tolerant while some are delay sensitive. The speed of the vehicles brings extra challenges for the RSUs to serve the vehicle requests because of dynamic changes in network topology and less time for connectivity between the vehicles and nearby RSUs. The infrastructure cost of RSUs does not allow the authorities to deploy them in large numbers to cover the entire city. This creates regions where vehicles cannot connect to any RSUs. The RSUs in road intersections are overloaded while some RSUs are far from the intersections with sufficient resources. This creates a mismanagement of available resources. This leads to an increase in delay, a decrease in the number of tasks completed, an increase in cost, and degraded QoS. The RSUs have limited capacities but the connectivity of RSUs with other nodes with available resources is not a constraint. Still, the service provided by the RSUs is not up to the mark. This poses an opportunity to work on request assignments for the RSUs. For request assignment, there is a need for a scheduler that can schedule the requests to the appropriate nodes by which the performance of the system is enhanced without affecting the QoS of the system.

1.2 Major Contributions of Thesis

This thesis contributes towards two major areas in vehicular networks: (1) Load balancing in RSUs, and (2) Profit maximization of service providers. The following sections discuss these contributions:

1.2.1 Contribution on Load Balancing

Load balancing is an important concept in the field of VANETs due to rapid changes in network topology and less time of interaction between the vehicles and the RSUs. The delay in receiving service requests is important as the vehicles move very fast. Getting the response after due time is not useful for some applications. The increase in applications in vehicles generates a huge amount of data that needs processing by some of the nodes. Although RSUs have more capacity in terms of resource availability, simultaneous requests from a huge number of vehicles make the RSUs overloaded. It becomes a challenge for the RSUs to allocate the request from the vehicles to suitable resources. Most of the proposed approaches do not consider many factors such as storage and computing resources simultaneously, and

continuous data generation while doing LB. The first contribution of this thesis is to propose a load-balancing algorithm considering the above factors. The following section explains the first contribution.

1.2.1.1 Application aware load balancing for road side units

In this work, our objective is to schedule VMs to RSUs in such a way that the number of VM requests that get served is maximized with minimum migration cost. In particular, we propose an algorithm called *Application Aware Load Balancing (AALB)* to balance the application load among RSUs while trying to meet the objectives stated above. AALB uses a Hungarian Matching algorithm to perform optimal allocation of VMs to RSUs. We have evaluated the performance of the proposed algorithm extensively through simulation using real traffic traces generated using SUMO (Simulation in Urban MObility) traffic generator [18]. The performance of AALB is compared with three other existing algorithms. The results show that AALB significantly improves the number of applications served and VM migration cost as compared to the three existing algorithms. The core contributions of our work are summarized below:

- We have formulated a scenario to schedule the requests of vehicles in RSUs that uses VM migration between RSUs to balance the load across RSUs. The objective is to maximize the number of VMs completed while minimizing the cost of VM migration.

- We have proposed an algorithm called *Application Aware Load Balancing (AALB)*, that efficiently allocates VMs to RSUs to increase the number of VMs completed while reducing the cost of VM migration.

- We have performed extensive simulations to evaluate the performance of AALB. In particular, we have compared the performance of AALB with three other existing algorithms with different types of applications to show that AALB performs significantly better.

It is important for a scheduler to choose a suitable RSU for task assignment, where the cost of migration and cost of renting the resources should be minimum without affecting the objectives. The following section explains the second contribution.

1.2.1.2 Resource renting for load balancing in VANETs

In this work, we assign the VM requests to RSUs such that a maximum number of VMs can execute with the least amount of time and cost. In particular, we have proposed an algorithm that we call Efficient Resource Renting (ERR) to meet the aforementioned objectives. Through simulations, we compared the effectiveness of ERR with two other known methods. The ERR is further modified to show the effectiveness of data partitioning by assigning requests to the RSUs. The results show a significant improvement with the proposed method as compared to the existing algorithms in terms of rent-out cost, VM completion percentage, total cost, and end-to-end delay. The following is a summary of this work's main contributions:

- To schedule the VMs' requests for RSUs, we have formulated our scenario as a weighted bipartite graph due to which problem becomes an assignment problem.
- We have proposed a pricing model to simulate the cost of renting out RSUs.
- We have proposed an algorithm that we call ERR to assign the VM requests to the RSUs by considering remaining storage capacity, remaining computing capacity, Data Processing Rate (DPR), and rent-out cost.
- Extensive simulations show the performance of ERR for periodic applications, event-driven applications, and their combinations. We found that ERR outperforms other existing algorithms in terms of average delay, average rent-out cost, VM completion percentage, and average total cost.
- We have analyzed the end-to-end delay of requests to measure the performance of ERR with consideration of data partitioning. We found that with data partitioning the assignment methods can be further improved.
- Simulation results show an improved performance of Modified ERR over other algorithms as well as ERR.

1.2.2 Contribution on Profit Maximization

RSUs have limited resources in VANETs. The increase in the number of requests by the applications running in the vehicles leads to the generation of a large number of service

requests. This leads to service failure and a decrease in QoS. Renting resources from other RSUs is considered one of the solutions. However, an increase in the number of applications and the generated data makes the available resources insufficient. Some of the resources of CC are not cost effective and accessing some resources increases the latency. Additionally, the infrastructure cost of RSUs is huge. Considering these scenarios, there is a need for a scheduling algorithm that can allocate the service requests of the vehicles to the appropriate node(CC/RSU/PV). The following section describes our third contribution.

1.2.2.1 Profit maximization in heterogeneous vehicular networks

The core contributions of this work are summarized below:

- We have formulated a problem to schedule the requests generated by the application of the vehicles to the nodes from RSUs, PVs, and CC.
- We have proposed an algorithm called Adaptive Assignment for Profit Maximization (AAPM) which maximizes the profit of the Service Provider (SP) and number of tasks completed.
- To show the cost associated with each node, a cost model is proposed. It helps in maximization of profit for the SP.
- We have performed extensive simulations to evaluate the performance of our proposed algorithm AAPM. We have compared the performance of AAPM with existing algorithms along with a GREEDY approach.

1.3 Organization of Thesis

This thesis is divided into six chapters based on the contributions mentioned above. The organization of the thesis is shown in Figure 1.3. The following is a list of the remaining chapters in this thesis.

- Introduction, motivation, and thesis contributions are presented in this chapter.
- In **Chapter 2**, the background and prior works related to the thesis contributions are discussed.

- In **Chapter 3** we present our first contribution. Here, we propose a load-balancing algorithm while considering the application types. We call the algorithm Application Aware Load Balancing (AALB). The application's type and deadline are considered while doing the assignment. AALB uses a Hungarian matching algorithm to perform optimal allocation of VMs to RSUs.

- **Chapter 4** details our second contribution, where we propose a resource renting algorithm that we call Efficient Resource Renting (ERR) for RSUs. This maximizes the number of VMs that finish running while minimizing total cost and average end-to-end delay. In addition to this, the effect of data partitioning is shown using analytical methods. To see the effect of data partitioning, the ERR is further modified while assigning requests to RSUs.

- **Chapter 5** presents our third contribution. Here, we propose an Adaptive Algorithm for Profit Maximization (AAPM) algorithm. The profit of the SP gets maximized without affecting the QoS. Here, the resources of other RSUs, PVs, and CC are rented while providing service to the users.

- **Chapter 6** concludes the thesis with a summary of the critical findings of the research. In addition, we also describe the future direction of the work conducted in the thesis.

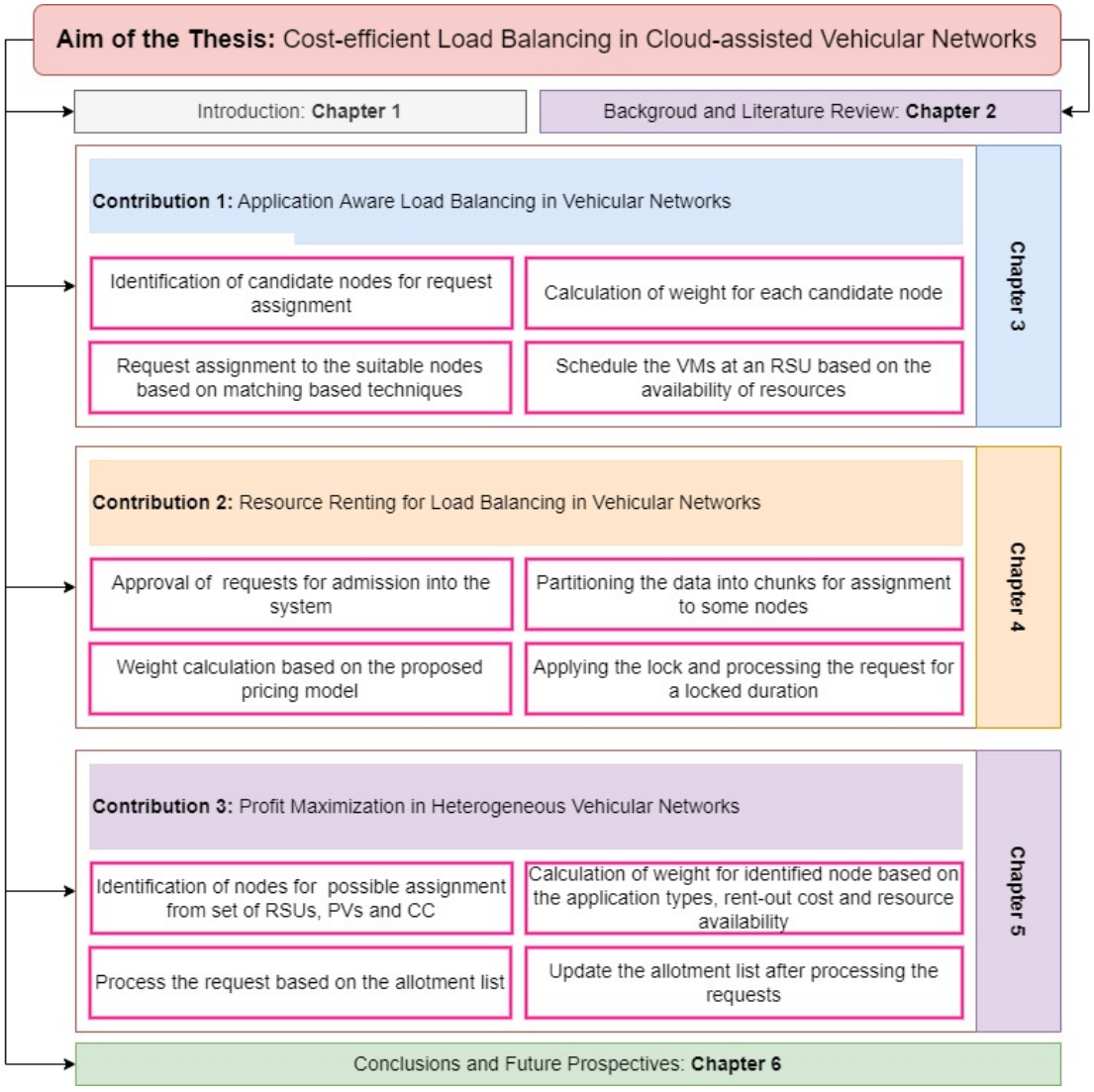


Figure 1.3: *Thesis organization*

2

Background and Literature Review

This chapter presents a brief overview of the background and literature review on load balancing. In the background study, we have presented the challenges, importance of load balancing, and finally listed some of the load-balancing techniques used in the VANET environment. Literature survey is broadly divided into two parts- 1) Load balancing techniques in VANETs and 2) Pricing Schemes in VANETs. Detailed works conducted on these topics by the researcher are given in Section 2.2.

2.1 Background

Load balancing at RSUs in VANETs is a critical aspect of optimizing network performance and resource utilization. As VANETs continue to gain prominence in the field of intelligent transportation systems, efficient management of network loads becomes of prime importance for ensuring reliable and seamless communication among vehicles and infrastructure. This background study provides an extensive overview of load-balancing techniques specifically used for RSUs in VANET environments. It explores the challenges associated with load distribution, the importance of load balancing, and existing methodologies in this domain.

2.1.1 Challenges in Load Balancing at RSUs

VANETs are characterized by rapidly changing traffic conditions which leads to fluctuating communication demands at RSUs and less time of connectivity between RSU and the vehicles. These RSUs handle diverse types of data, including safety-critical messages, multimedia content, and internet connectivity requests. RSUs typically have constrained processing power, memory, and bandwidth, necessitating efficient allocation and utilization of resources. High-density vehicular environments can result in channel interference and congestion, impacting the performance of RSUs.

2.1.2 Importance of Load Balancing

Load balancing mitigates the risk of network congestion and reduces the likelihood of packet loss, enhancing the reliability of communication in VANETs. By distributing the workload evenly across RSUs, load balancing helps to maintain consistent QoS metrics such as latency, throughput, and jitter. Efficient load balancing maximizes the utilization of RSU resources, minimizing idle capacity and ensuring optimal network performance. Load balancing mechanisms facilitate the scalability of VANETs by adapting to varying traffic loads and network conditions, thus accommodating future growth.

2.1.3 Existing Load Balancing Techniques

There are various types of LB techniques used in the VANET:

- Traffic-Aware Routing Algorithms: These algorithms consider traffic conditions and network topology to dynamically route data packets towards less congested RSUs [19].
- Dynamic Resource Allocation: Here, RSUs dynamically allocate resources such as bandwidth and processing power based on real-time demand, ensuring fair distribution among vehicles [20].
- QoS-based Load Balancing: These algorithms prioritize traffic based on QoS requirements, allocating resources accordingly to meet Service Level Agreements (SLAs) for different applications [21].
- Centralized vs. Distributed Approaches: Load balancing can be performed centrally, where a central controller manages resource allocation, or in a distributed manner where RSUs cooperate to balance the load autonomously.

Traffic aware load balancing focuses on the number of vehicle in a specified region. However, the applications and their resource requirement is an important aspect while doing load balancing. This is being overcome by the dynamic resource allocation methods. QoS-based load balancing only focuses on service quality without considering the number of requests processed within the deadline.

2.2 Literature Review

In this Section, we have focused on two important aspects - 1) Load balancing in VANETs and 2) Resource renting in VANETs.

2.2.1 Load balancing in VANETs

Load balancing plays a critical role in VANETs due to the high-speed mobility of vehicles and the constantly changing network topology. The generated data by these vehicles could not be processed by the vehicles and even by the RSUs in an overloaded scenario. This is because of higher vehicle density and a huge amount of data generation by some of the applications running in the vehicles.

In [22], authors have proposed a cooperative load balancing algorithm where they have used the direction of vehicles, load at destination RSUs, and delay bound of request as the factors for load balancing. Here, as RSUs near the destination are selected for offloading, it may lead to overloaded scenarios when many vehicles intend to reach a common destination. In [23], a dynamic load balancing algorithm is proposed by partitioning the total number of RSUs into three categories– light-load, normal-load, and heavy-load. A new task is assigned to a relatively less loaded RSU. Here, due to the mobility of vehicles, the RSU sets belonging to a particular group may change frequently. While their algorithm includes VM migration, they haven't addressed its impact from a load-balancing perspective. In [24], authors proposed k-Shortest Paths between each source and destination willing to have communication with the use of the cost model proposed. When forwarding packets, the algorithm distributes the load between all intermediate nodes by choosing the next hop according to road conditions, either to nodes with the same moving direction or those with lower collision probability. During the connections, the protocol warns about path congestions when confronting prior congestion threshold. In such cases, related connections could switch to less congested paths. As a result, the traffic load will be balanced throughout

the entire network and the approach will gain a higher packet delivery ratio and throughput. Chi-Fu et al. [25] have proposed two algorithms for load balancing in a VANET scenario. The first algorithm partitions the scenario into sub-regions based on the RSUs' location. RSUs provide services to vehicles in their regions. An RSU provides Internet access for vehicles in its sub-region and the boundaries between sub-regions change dynamically to adapt to load and density changes. However, the authors do not address the cost of VM migration. In the second algorithm, for offloading, they have proposed the selection of an RSU from the group of vehicles. Here, the presence of coverage holes may significantly affect the performance.

The work in [26] has considered a heterogeneous vehicular environment with variable storage and computation capacity of vehicles. Their proposed algorithm provides service partially by the Vehicular Cloud (VC) and partially by the Remote Cloud (RC). In the overloaded scenario, the extra load is transferred from VC to RC. As the location of the VC may be geographically far from the RC, it may induce a further delay in the execution of the application. In [27], authors have considered a load-aware offloading scheme. They have assumed that the load at each RSU will be known to all vehicles and the resource requests will be offloaded to RSUs with minimum load. They have also assumed multi-hop communication between vehicles which may introduce more delay in servicing the requests. In [28], authors have considered both fiber and wireless mediums to offload data from vehicles. Data is offloaded to one of the RSUs or the remote cloud based on the delay bound. Game theoretic approach is used to decide the data offloading path. Here, the focus is on minimization of processing delay of VMs. The authors have considered overlapping in RSU's transmission range, but the effect has not been discussed in the process of RSU selection. In [29], authors have proposed a contract-based offloading scheme while considering the delay bound of applications. They have also taken the price of each unit of computation into consideration. However, in this work, the authors only focus on the computation resource and the significance of available storage on the execution of applications has been ignored too.

In [30], the authors have proposed a Joint algorithm for Selection decision, Computation resource, and Offloading (JSCO). Here, a task/application request is partitioned into two parts - one part is executed by the vehicle locally and the other part is executed in the RSU. The JSCO algorithm selects an RSU for offloading tasks based on its available computation capacity and the number of vehicles in its region. The focus of the authors is only on the

computation capacity of RSUs. They have ignored the effect of available storage at RSUs and do not address VM migrations and the corresponding cost incurred. In [31], authors proposed one algorithm for selection of one computation unit from three different options. The options are local vehicles, RSUs, and central cloud. The decision is made based on the utilization value of the vehicle, RSU, and central cloud. The proposed algorithm is termed Distributed Computation Offloading and Resource Allocation (DCORA). Although the combination of cloud and MEC servers is used to schedule the task, its effect on the number of tasks completed has not been discussed. The work in [32] focuses on dependencies between tasks while doing the load balancing. In this work, the authors proposed a Multiple Applications Multiple Tasks Scheduling (MAMTS) algorithm to solve the assignment issue. Each application is modelled as a directed acyclic graph. Applications consist of several tasks and some tasks are dependent on other tasks. Multiple applications and multiple tasks are prioritized to minimize the completion time of applications. The overloaded RSU scenario is not considered in this work.

2.2.2 Resource Renting in VANETs

Here, we discuss the works that focus on resource renting. The pricing scheme is discussed below.

Cloud computing brings the novel market-oriented pay-as-go model which makes it convenient for many users to use different types of resources without bothering about the server's infrastructure. The user only pays the rent-out cost for the resources and uses the resources without any interruption [33]. Using this technique a number of researchers have worked and proposed some pricing schemes for different scenarios. In [34], a cloud federation formation framework is modelled for the case, when a request is made by a user to the cloud broker, consisting of the requirement of a number of computing resources and a preferred individual service provider through which the users seek to get resource services. Authors in the work [35], present a cloud system model for the cloud provider to dynamically expand the scale of geo-distributed data centers.

In [3], authors have proposed a dynamic resource pricing model for sharing resources between the cloud providers. Here, the proposed algorithm tries to minimize the cost of the vehicular service provider while meeting the delay bounds of different vehicular applications with fewer VM migrations. Both cost and delay are considered while classifying the user's

request for processing. In the study [36], authors focused on maximizing profit for cloud brokers through optimal multiserver configuration and resource pricing. They treated the cloud broker as a multiserver system and employed an M/M/n/n queuing model to analyze various factors influencing profitability. The analysis included an examination of the relationship between the sales price of VMs and customer demand, leading to the calculation of the expected charge for a VM request. Through a series of numerical calculations, the authors demonstrated that the cloud broker could effectively reduce costs for cloud users while maintaining a significant level of profitability.

Authors in [37] introduce a novel approach to virtual resource renting, aiming to adaptively refine the rental strategy based on both price distribution and task urgency. By factoring in task urgency and price distribution, the authors devise a weak equilibrium operator to compute the acceptable price for each virtual resource type. Virtual resources meeting the acceptable price criteria are aggregated into a set. Subsequently, a price prediction algorithm is proposed to forecast the next price interval for these virtual resources. Lastly, an innovative rental decision-making algorithm is formulated to identify the most profitable resource from the set. In paper [38], an algorithm is proposed that enables vehicles to discover and consume services of mobile cloud servers that are moving nearby. Public buses spread service registration information to the buses within the same bus line, and to buses of connected lines, as well. A consumer vehicle can discover the available services, along with their constraints, by querying the buses in their vicinity. To ensure service consumption, the buses provide a routing protocol allowing communication between the provider and consumer. Then the algorithm is extended. On the one hand, vehicle providers select the most appropriate public bus for efficient service registration, and users select the most satisfactory service, which satisfies both provider constraints and user preferences. The cost of renting resources is not focused on while selecting the destination node.

In the era of beyond 5G technologies, deploying deep neural networks (DNNs) on IoT devices is challenging due to limited computational resources. To address this, researchers have proposed a split computing approach, where DNN inference is divided between the IoT device and a nearby edge device. Key mechanisms supporting this approach include Dynamic Split Computation (DSC) for optimal task partitioning, Reliable Communication Network Switching (RCNS) for selecting the best available network, and Task Load Balanc-

ing with Prioritization (TLBP) to manage workload distribution and device limitations such as battery life and resource contention. Experimental results demonstrate that this approach significantly reduces inference time and enhances system efficiency, making it suitable for real-time IoT applications [39]. The rapid increase in mobile traffic and uneven distribution across bands in multi-band networks can lead to congestion and degraded user experience. Traditional load balancing methods often rely solely on channel quality, overlooking user demands and band resource availability, which results in inefficient resource utilization. To overcome this, an event-based load balancing algorithm has been proposed, modeling the problem as a multi-objective stochastic optimization. It assigns user equipment (UE) to bands probabilistically, aiming to balance traffic while minimizing inter-frequency handovers. Simulations show improved throughput and reduced interruption time compared to traditional approaches [40].

The data generation rate of applications running on vehicles can be periodic or event-driven. The existing approaches in the literature do not focus on this aspect of data generation. In this thesis work, we have considered both periodic and event-driven data generation from vehicles. Additionally, we have considered continuous data generation from vehicles while performing load balancing.

Table 2.1: *Existing literature*

Reference	RSU Storage	RSU Computation	Continuous data generation	Approach Used
[22]	✓	✗	✗	Direction of vehicles is used to assign load
[41]	✓	✗	✗	New request is assigned to relatively less loaded RSUs
[42]	✓	✗	✗	Find the lowest congested path between source and destination
[43]	✓	✗	✗	Transmission range of RSU is adapted based on the load
[26]	✓	✓	✗	Heterogeneous scenario with processing in vehicular cloud and central cloud
[27]	✗	✓	✗	Multi-hop communication between RSUs
[29]	✗	✓	✗	Contract-based uploading scheme is used
[44]	✗	✓	✗	Partition tasks into two parts, one is executed locally and the other in RSU
[31]	✗	✓	✗	Select one of the options from vehicles, RSU, and central cloud based in utilization value
[32]	✗	✓	✗	Dependency among the tasks are focused while scheduling
Proposed Approach	✓	✓	✓	Matching based assignment of VMs to RSUs based on applications types

3

Application Aware Load Balancing in Vehicular Networks

3.1 Introduction

In VANETs, LB refers to the efficient distribution of network traffic, communication tasks, and computational tasks among the vehicles or nodes in the network [45]. VANETs are a specific type of mobile ad-hoc network where vehicles communicate with each other and with roadside infrastructure to enhance road safety, traffic management, and provide various services. LB is crucial in VANETs to ensure optimal utilization of resources and to avoid network congestion or uneven distribution of communication loads among vehicles [46]. LB includes the dissemination of safety messages, traffic updates, and other information. Uneven distribution could lead to congestion in certain areas and delays in message delivery. Vehicles in a VANET generate and exchange messages for various purposes, such as collision warnings, traffic information, and location updates. Load balancing can happen for vehicles and RSUs. This chapter of the thesis focuses on load balancing in RSUs.

LB ensures that no single RSU or subset of RSUs is overloaded with communication tasks, preventing bottlenecks, and improving overall network performance. LB optimizes

the utilization of available resources, including bandwidth and processing power, among the vehicles. Efficient use of resources helps in reducing communication delays, enhancing the reliability of information dissemination, and maximizing the overall capacity of the VANET [47]. VANETs are dynamic and highly variable in terms of network topology and communication patterns. LB mechanisms need to adapt to these changes in real-time. Dynamic LB algorithms can adjust the distribution of tasks based on the current network conditions and the mobility of vehicles. LB plays a crucial role in maintaining QoS in VANETs. It ensures that the communication quality, in terms of latency, reliability, and availability, meets the requirements of safety-critical applications, such as collision avoidance systems. LB can be achieved through centralized or decentralized approaches. In a centralized approach, a central entity coordinates the LB decisions for all RSUs. In a decentralized approach, RSUs collaboratively make LB decisions based on local information. The primary causes of the overloading scenario are huge data generation by the applications in the vehicles and an increase in the number of simultaneous service requests by the vehicles.

In VANET, vehicles are equipped with On-Board Units (OBUs) that can communicate with RSUs placed at various locations along the roads to enable different types of applications. Such applications can be categorized as safety applications, public service applications, location-based services, multimedia, and entertainment applications, etc. [8]. Applications can generate data either in a periodic manner or on the occurrence of an event. Some applications that generate data periodically are navigation systems, safety warnings, lane change, etc. [48, 49]. Online gaming, VoIP, multimedia, parking slot locator, etc. are some examples of applications that generate event-driven data [49, 50]. Many of these applications can generate a large amount of data that may need to be processed. RSUs can be equipped with computing and storage resources to process such data. The data generated may also have local relevance in terms of space, time, and user interest [51].

3.2 System Model

Our system model in the proposed work consists of vehicles, RSUs, and the CC as shown in Figure 3.1. RSUs are connected via a wired connection. Each vehicle runs applications $(A_1, A_2, ...A_k)$ according to the users' interest as shown in Figure 3.2. Each application in a vehicle has a different data generation rate. The amount of data generated by each

vehicle at each time step is known. In our system model, we have assumed that each vehicle generates requests for only one application at a time, and correspondingly, only one VM is created in the RSU for the vehicle, i.e., 1 vehicle=1 application= 1VM. This asumption is taken because, if a VM runs multiple applications then the migration of VMs will affect the system's performance due to task dependencies. Authors in [52] have considered such a system. The amount of data generated by the applications is stored in the originating vehicles temporarily. When a vehicle is within the transmission range of an RSU, it sends a connection request to the RSU. A connection request is accepted based on availability of channels and storage. In our work, we assume that the first instance of a vehicle's request, corresponding to a given application, creates a VM in an RSU depending on the availability of resources. In subsequent instances, the request from the vehicle means getting service from the VM already created in the RSU. There is a fixed number of channels that are used by vehicles to transmit the data based on the IEEE 802.11p standard [53]. When no channel is free, the data is held with the vehicle until the next time instant when it comes in the transmission range of an RSU. This may lead to an increase in delay and/or deadline miss of a task. The data present in the vehicles are transferred to one RSU in the transmission range. A VM is created at the RSU to process the data generated by the applications running in the vehicles. Each application has a lifetime and the data to be processed within the lifetime of the application. The information about the lifetime of the application is communicated to the RSU along with the request sent by a vehicle.

In this work, connecting the local server to RSUs means specifying that RSUs are not simple interfaces. These RSUs are capable of storage and computing units. The connected RSUs can be called edge clouds. The requirement of an edge cloud is there because of the limited capacity of the edge node which is an RSU in our scenario. These resources are used by the VMs running for different applications. A Load Balancing Client (LBClient) is a software module that runs at RSUs. LBClient is a part of the load balancer that accepts or rejects the connection request from the vehicles as shown in Figure 3.2. The transmission range of RSUs is non-overlapping. For each vehicle, one VM is created at some RSU which can be migrated to another RSU to meet the objectives and constraints. At any time, a VM resides in exactly one RSU and occupies some amount of storage that depends on the application type and the amount of unprocessed data. If a VM request is not accepted at a particular time instant, an attempt is made to process it in the next time instants. The Load Balancing Server (LBServer) running in the CC has information about all the

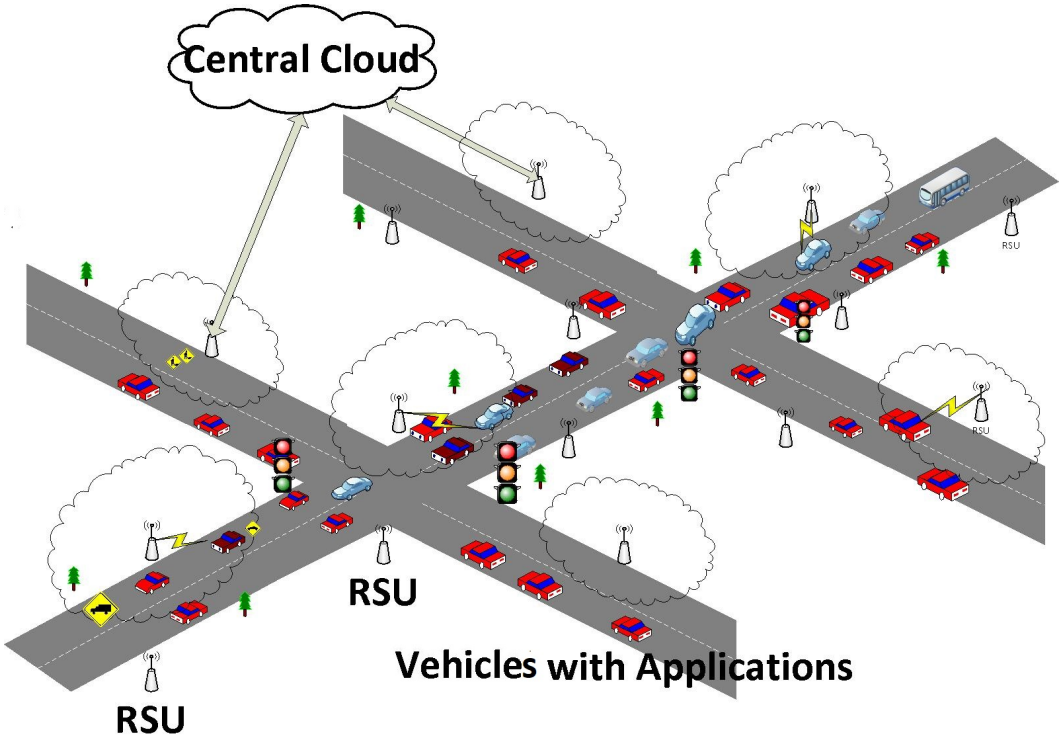


Figure 3.1: *System model*

RSUs and makes decisions based on the information present in all RSUs. The storage and computation resources used by LBServer and LBClient are negligible. Therefore, we have ignored this in the current work. A VM can be present in an RSU without being scheduled when the available computing resources of that RSU are less than the required computation resources. At that time only the storage resources of RSU are used and not the computing resources. The time taken to execute a task depends on the amount of unprocessed data and the available computing capacity of the RSU to which it is assigned. In the case of VM migration, the available unprocessed data is migrated from one RSU to another. The migration incurs some amount of cost called migration cost. This migration cost has two parts - a fixed cost and a variable cost. The fixed cost is the cost of migrating a VM from one RSU to another RSU. It is the same for all migrations in the system. The variable cost is calculated based on the amount of unprocessed data to be migrated and the cost per unit of data already migrated. In our algorithm, a VM can migrate to any RSU in the network, irrespective of whether the RSU is in the route of the vehicle or not. A VM remains alive until the corresponding vehicle departs from the last RSU in its route.

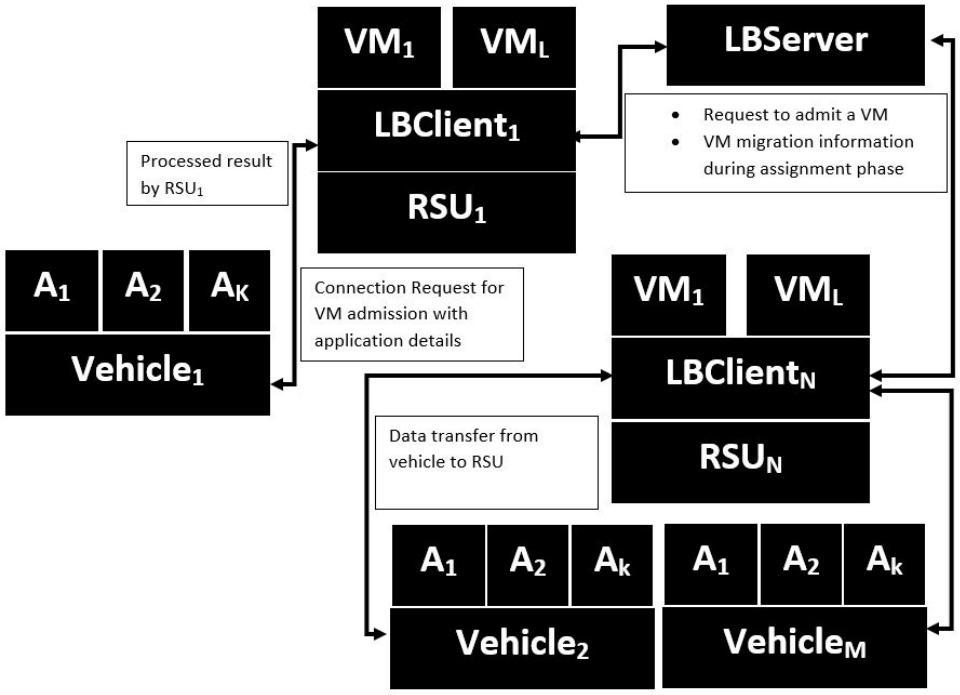


Figure 3.2: *Request assignment diagram*

3.3 Problem Formulation

In this Section, we present the problem addressed in this paper formally as an optimization problem. In particular, we specify the input variables, the output variables, a set of variables whose values are derived from the input and output variables, the constraints, and the objective functions.

3.3.1 Assumptions

1. The coverage region of any two RSUs does not overlap.
2. The route and speed of each vehicle are known a-priori.
3. The data generation pattern of the vehicles (how much data is generated at each time instant by a vehicle) is known a-priori.
4. Data from a vehicle can only be transferred to an RSU when the vehicle enters into the coverage range of that RSU.

Table 3.1: *Table of symbols*

Symbol	Description
T	Total duration of travel of all the vehicles
X	Total number of vehicles
Y	Total number of RSUs
R	Set of RSUs
p_i^r	Maximum computing power of RSU r_i
s_i^r	Maximum storage capacity of RSU r_i
D^r	Number of units of data that can be processed by RSU r_i per unit time (same for all RSUs)
$Count_r$	Maximum number of channels available for data transfer between a vehicle and an RSU
V	Set of vehicles
$start_j$	Start time of the journey for vehicle v_j
P_j^v	Path followed by vehicle v_j
a^j	Application being run by vehicle v_j
$\mathcal{L}_j^v$	Lifetime of the application for v_j ($\mathcal{L}_j \leq T$)
Λ_j^v	Rate of data generation by a vehicle v_j
A	Set of applications
p_k^a	Amount of computing power needed by application a_k
s_{kf}^a	Fixed amount of storage required for the application a_k(independent of user's data)
$Cost_m^f$	Fixed migration cost for application a_k
$Cost_m$	Migration cost for one unit of data (same for all applications)
x_{ijt}	Output variable. Its value is 1, if a VM for a vehicle v_j is present in an RSU r_i at time t, 0 otherwise
y_{ijt}	Output variable. Its value is 1, if a VM for a vehicle v_j is scheduled in an RSU r_i at time t, 0 otherwise
T_j^i	Duration for which vehicle v_j remains in the transmission range of RSU r_i
$D_{left_j}^t$	Amount of data left in vehicle v_j at the beginning of time instant t
U_t^j	Quantity of unprocessed data in the RSUs for a vehicle v_j's VM at time t
C_j	Indicator variable whose value is set to 1 if the amount of unprocessed data of the vehicle v_j is 0 when the vehicle leaves the last RSU in its path, and 0 otherwise
I_j^t	Indicator variable whose value is 1 if vehicle v_j has migrated from one RSU to another at time t, 0 otherwise

3.3.2 Variable Declaration

In this Subsection, we define three types of variables – input variables, output variables, and derived variables in detail. All variables are also listed in the table of symbols given in Table 3.1.

3.3.2.1 Input Variables

I. Let the total duration of travel of all the vehicles be T, with the time slots denoted by $1, 2, \ldots, T$.

II. Let $R = \{r_1, r_2, \ldots r_Y\}$ be the set of RSUs. Hence, the number of RSUs is Y. Each $r_i \in R$ is represented by the tuple $\langle p_i^r, s_i^r, D^r \rangle$ where

- p_i^r = Maximum computing power of r_i
- s_i^r = Maximum storage capacity of r_i
- D^r = Number of units of data that can be processed by r_i per unit time (same for all RSUs)

III. Let $Count_r$ be the maximum number of channels available for data transfer between a vehicle and an RSU. Hence, it is also the maximum number of requests that can be accepted by one RSU at a particular time instant.

IV. Let $V = \{v_1, v_2, \ldots, v_X\}$ be the set of vehicles where X is the number of vehicles. Hence, the maximum number of VMs that can be present in the system at any time is X, as each vehicle runs at most one application. Each vehicle $v_j \in V$ is represented by the tuple $\langle start_j, P_j^v, a^j, \mathcal{L}_j^v, \Lambda_j^v \rangle$ where

- $start_j$ is the start time of the journey for v_j
- P_j^v is the path followed by v_j, represented as the sequence of RSUs $\langle r_1^j, r_2^j, \ldots r_{(K^j)}^j \rangle$ in the path before the lifetime of the application running in the vehicle v_j, where K^j is the number of RSUs in $|P_j^v|$. For each RSU r_k^j, the arrival and departure time of v_j at r_k^j is denoted by $arr_{r_k}^j$ and $dep_{r_k}^j$ respectively
- a^j is the application being run by v_j
- $\mathcal{L}_j^v$ denotes the lifetime of the application for v_j ($\mathcal{L}_j \leq T$)

- Λ_j^v is the rate of data generation by a vehicle v_j represented by the sequence $\langle \lambda_1^{v_j}, \lambda_2^{v_j}, \ldots, \lambda_T^{v_j} \rangle$ where $\lambda_k^{v_j}$ denotes the amount of data generated by v_j at time slot k. The data generated after arrival at last RSU cannot be processed as there is no other RSU in the route after that for the vehicle to get back the result of the processing. Therefore, it is considered to be zero for all time instants after the arrival at the last RSU. The data generated may be zero at some time instant between start of the application and the lifetime of it. Also, as it is assumed that one vehicle can run only one application at a time, the maximum number of VMs that can be present in the system at any time is X. In this thesis, we refer to the terms *lifetime of an application* and *lifetime of vehicle* interchangeably.

V. Let $A = \{a_1, a_2, \ldots a_k\}$ denote the set of applications. Each application $a_k \in A$ is represented by the tuple $\langle p_k^a, s_{kf}^a, t_{kf}^a, Cost_m^f \rangle$, where

- p_k^a = amount of processing/computing power needed by a_k
- s_{kf}^a = fixed amount of storage required for the application a_k(independent of user's data)
- $Cost_m^f$ = fixed migration cost for a_k
- $Cost_m$ = migration cost for one unit of data (same for all applications)

3.3.2.2 Output Variables

- x_{ijt} for all $r_i \in R$, for all $v_j \in V$ and for all $t, 1 \leq t \leq T$
  - $x_{ijt} = 1$, if a VM for a vehicle v_j is present in an RSU r_i at time t
  - $x_{ijt} = 0$, if a VM for a vehicle v_j is not present in an RSU r_i at time t.
- y_{ijt} for all $r_i \in R$, for all $v_j \in V$ and for all $t, 1 \leq t \leq T$
  - $y_{ijt} = 1$, if a VM for a vehicle v_j is scheduled in an RSU r_i at time t‘
  - $y_{ijt} = 0$, if a VM for a vehicle v_j is not scheduled in an RSU r_i at time t.

3.3.3 Derived Variables

In this Subsection, we refer to a set of variables that are derived from the input and output variables. These are listed below.

- $\mathfrak{T}_j^i = dep_{r_i}^j - arr_{r_i}^j$ is the duration for which vehicle v_j remains in the transmission range of RSU r_i.
- $D_{left_j}^t$ is the amount of data left in vehicle v_j at the beginning of time instant t.

$$D_{left_j}^t = \begin{cases} 0, & \text{if (C } \wedge \text{ E) is true} \\ (D_{left_j}^{t-1} + \lambda_{t-1}^{v_j}) - (T_r^{t-1} \times t_r), & \text{if (A } \wedge \text{ C } \wedge \text{ F) is true} \\ (D_{left_j}^{t-1} + \lambda_{t-1}^{v_j}), & \text{if (B } \vee \text{ D) is true} \end{cases}$$

where $\lambda_t^{v_j}$ denotes the amount of data generated by v_j at time slot t and T_r^{t-1} is the amount of data transferred at time instant $t-1$, and A, B, C, D, E, & F denote conditions defined as follows:

  - A- Vehicle v_j is in the transmission range of r_i.
  - B- Vehicle v_j is not in the transmission range of r_i.
  - C: Channel is available for vehicle v_j to transmit data.
  - D: No channel is available for vehicle v_j to transmit data.
  - E: $\frac{D_{left_j}^{t-1} + \lambda_{t-1}^{v_j}}{t_r} \leq (\mathfrak{T}_j^i)$, time required to transmit the amount of data left in the vehicle is less than or equal to the duration for which vehicle v_j remains in the transmission range of RSU r_i.
  - F: $\frac{D_{left_j}^{t-1} + \lambda_{t-1}^{v_j}}{t_r} > (\mathfrak{T}_j^i)$, time required to transmit the amount of data left in the vehicle is greater than the duration for which vehicle v_j remains in the transmission range of RSU r_i.

- Let U_t^j be the amount of unprocessed data in the RSUs for the VM of a vehicle v_j at time t. It is the difference between the data transferred to the RSUs by the vehicle till time t and the data processed by the RSUs till time t. The amount of data transferred is the difference of data generated and remaining data in the vehicle till time t. The amount of data processed is equal to the sum of all the data processed by each RSU till t. The data processed in an RSU can be calculated by multiplying the rate of data processing by RSU with number of time instants at which the VM was scheduled.

$$U_t^j = (\Sigma_{k=1}^{arr_{r_c}^j} \lambda_k^{v_j} - D_{left_j}^t) - D^r \times \Sigma_{i=1}^{Y} \Sigma_{t=1}^{T} y_{ijt} \tag{3.1}$$

where r_c is the last RSU in the vehicle's path before t. Let C_j be an indicator variable whose value is set to 1 if the amount of unprocessed data of the vehicle v_j is 0 when the vehicle leaves the last RSU in its path and 0 otherwise. Similarly, let I_j^t be an indicator variable whose value is set to 1 if $(x_{i_1jt} = 1 \wedge x_{i_2jt_1} = 1 \wedge t_1 = t{+}1 \wedge i_1 \neq i_2)$, 0 otherwise. Thus, $I_j^t = 1$ indicates that the vehicle v_j has migrated from one RSU to another at time t.

3.3.4 Objective Function

The problem we address is to maximize the number of VMs completed while minimizing the cost of VM migration. Hence, the objective functions are stated as

$$maximize \;\; \Sigma_{j=1}^{X} C_j \tag{3.2}$$

$$minimize \;\; \Sigma_{j=1}^{X} \Sigma_{t=1}^{T} I_j^t \times \left(U_t^j \times Cost_m + Cost_m^f \right) \tag{3.3}$$

3.3.5 Constraints

- A vehicle v_j should not generate any data after the arrival at the last RSU, r_l^j, in its route.

$$\lambda_t^v = 0, \;\; \forall v_j \in V, \; \forall t > arr_{r_l}^j \tag{3.4}$$

  where r_l^j is the last RSU that the vehicle v_j passes by before $\mathcal{L}_j^v$.

- VM will not be created for a vehicle until its arrival time at the first RSU in its route.

$$x_{ijt} = 0, \;\; \forall v_j \in V, \; \forall t, \; 1 \leq t < arr_{r_1}^j \tag{3.5}$$

- For any vehicle v_j, at any time t between the arrival time of v_j at the first RSU r_1^j and the departure time of v_j from the last RSU r_l^j in its route, the VM for v_j must be present in exactly one RSU.

$$\Sigma_{i=1}^{Y} x_{ijt} = 1, \;\; \forall v_j \in V, \; \forall t, \; arr_{r_1}^j \leq t \leq dep_{r_l}^j \tag{3.6}$$

- A VM will be scheduled only when it is present at an RSU and there is some amount of unprocessed data in the VM.

$$y_{ijt} = 1 \implies (x_{ijt} = 1 \wedge U_t^j \neq 0) \tag{3.7}$$

- At any time t, the total computing capacity needed by all the VMs scheduled at an RSU should not exceed the total computing capacity of the RSU.

$$\sum_{j=1}^{A} p_{aj}^{a} \times y_{ijt} \ \leq p_i^r, \ \ \forall r_i \in R, \ \forall t, 1 \leq t \leq T \tag{3.8}$$

- At any time t, the total storage needed by all VMs present at an RSU should not exceed the storage capacity of the RSU.

$$(\sum_{j=1}^{X} U_t^j + \sum_{j=1}^{X} s_{k_f}^{a^j}) \times x_{ijt} \ \leq s_i^r, \ \ \forall r_i \in R, \ \forall t, 1 \leq t \leq T \tag{3.9}$$

- A VM must be scheduled in only one contiguous time block at an RSU.

$$\begin{aligned}((y_{i_1jt_1} = 1) \wedge (y_{i_1jt_3} = 1) \wedge \nexists\, i_2 \neq i_1, y_{i_2jt_2} = 1) \implies \\ \forall t, t_1 \leq t \leq t_3, \ \ y_{i_1jt} = 1, \ \ \forall v_j \in V, \ \ \exists t_1, t_2, t_3, 1 \leq t_1 < t_2 < t_3\end{aligned} \tag{3.10}$$

- Total execution time required by a vehicle's application should be less than the departure time of the vehicle from the last RSU, r_l^j, in its route.

$$(\Sigma_{t=1}^{arr_{r_l}^j} \lambda_t^j)/(D^r \times \Sigma_{i=1}^{Y} \Sigma_{t=1}^{dep_{r_l}^j} y_{ijt}) \ \leq \ dep_{r_l}^j, \ \ \forall v_j \in V \tag{3.11}$$

The constraints are formed to based on the real life problem scenario. The vehicle may generate data any time throughout its journey. However, the data generated beyond the transmission of last RSU of its path can not be processed in the given scenario. To address that first constraint is defined. A VM is only created if vehicle has generated some data and its request to process is accepted by the RSU. To make the scenario realistic the second assumption is defined. During the VM migration the a VM migrates from one RSU to other. However, it can not considered as presence in multiple RSUs. To make the scenario realistic third assumption is defined. Similarly in constraints fifth and sixth, it is defined that the total need of the computing and storage resources should not exceed the total availability of the resources. In real-life use of application if processing time is greater than the expected deadline the output of the processing becomes meaningless for the user. To bring this idea to the problem scenario, the assumption eighth is defined.

3.4 AALB: Application Aware Load Balancing

In this Section, we propose an algorithm that we call Application Aware Load Balancing (AALB) where the major focus is to schedule the VM requests based on the data generation rate of the applications. Application type is identified by its criticality and robustness. The criticality is measured by its deadline and the robustness is measured by the data generation rate of application. It is assumed that One task is generated per vehicle and the task whose deadline is near gets a chance to execute before the task whose deadline is far. The proposed AALB algorithm invokes a sequence of modules at each time step to assign VMs to RSUs.

AALB considers the state of RSUs and VMs in the last time step to make decisions in the current time step. The basic working model of AALB consists of three modules – *AdmissionControl*, *VMAssignment*, and *VMScheduled*.

1. *AdmissionControl*: The *AdmissionControl* module is the entry point of VMs to the system. It updates the amount of data required by the VM based on the type of application. It checks whether a VM can be assigned to at least one RSU in the system or not. If such an RSU is found, the VM is admitted to the system, otherwise, the VM tries to enter the system in the next time step. The data generated by the vehicle may not get a chance to be uploaded to the RSU fully. In that case, the rest of the data is transferred in subsequent time instants.

2. *VMAssignment*: It chooses a set of VMs to be considered for assignment in the current time step and selects the RSUs to which the VMs will be assigned using the Hungarian Matching algorithm. Note that, this can change the current assignment of a VM and cause it to migrate to some other RSU.

3. *VMScheduling*: It checks for availability of computing resources for VMs assigned to an RSU, and chooses the subset of VMs to be scheduled. The scheduled VMs are then executed and their status is updated.

For *VMAssignment*, our system scenario can be considered as a bipartite graph, with one partition representing the set of VMs and the other representing the set of RSUs, as shown in Figure 3.3. Edges are assigned between a VM and an RSU depending on whether the VM can be assigned to the RSU. Weights are assigned on each edge based on a combination of two factors – migration cost and the time RSUs take to execute the VMs. A

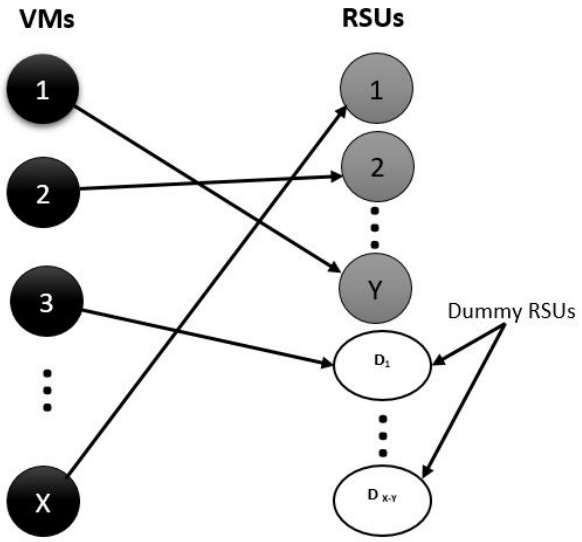


Figure 3.3: *VM to RSU assignment*

matching algorithm is then run iteratively on this bipartite graph to find the assignment of VMs to RSUs. We use the Hungarian Matching algorithm (also called Kuhn-Munkres algorithm) [54] for finding the maximum-weight (minimum-weight) matching. The algorithm works for a balanced problem where the number of nodes in both sets is equal. In our scenario, the number of VMs may be more than that of RSUs. In such a case, some dummy RSUs with zero resources are introduced into the system to make the number of RSUs equal to the number of VMs, as shown in Figure 3.3. Assignments made to dummy RSUs can be discarded at the end, and the corresponding VMs can be considered for assignment again. The motivation for selecting the Hungarian Matching algorithm for our work is as follows. In this work, we consider an optimization problem which corresponds to a matching problem, where a set of VMs need to be matched with a set of RSUs. The potentially large number of vehicles and RSUs makes the scenario a large-scale assignment problem. Hungarian matching algorithm is known to work better for such large-scale assignment problems [55]. One example application can be found in [56] that assigns several mobile stations to a large number of distributed single-antenna access points. Therefore, we have also been motivated to use the Hungarian Matching algorithm for assignment of VMs to RSUs.

The pseudocode for the overall execution of the AALB algorithm is shown in Algorithm 1. The algorithm calls three modules – *AdmissionControl*, *VMAssignment*, and *VMScheduled* in each time step up to the final time T.

Algorithm 1: Load Balancing

```
1 R_t ← Array of RSUs at time instant t
2 for (t = 0; t < T; t = t + 1) do
3     AdmissionControl
4     VMAssignment
5     VMScheduled
```

Next, we describe the data structures used in these three modules along with a brief description of the modules.

3.4.1 Data Structures Used

At each time step t, AALB algorithm keeps two primary data structures, an array R_t for storing the status of each RSU at time t, and an array M_t for storing the status of all existing VMs at time t. Each element $R_t[i]$ stores the following information:

- Identification number *id* of the RSU
- List of VMs v that are assigned to the RSU
- List of VMs v_s that are scheduled in the RSU
- The remaining storage capacity $remaining_s$ of the RSU
- The remaining computation capacity $remaining_c$ of the RSU
- A binary variable *rsu_scheduled* that is set to 1 if at least one VM is scheduled in the RSU at time $t - 1$.

Similarly, each element of $M_t[j]$ stores following information:

- Identification number *id* of the VM
- Storage needs s of the VM

- Computing needs c of the VM
- The RSU r to which the VM is assigned to at the beginning of time t
- A binary variable *scheduled* which is set to 1 if the VM is scheduled at time t
- The amount of new data *data_amount* is generated for the VM at the beginning of time t
- The remaining amount of data *data_remaining* is to be processed at the beginning of time t
- A binary variable *data* which is set to 1 if new data is there for the VM at the beginning of time t
- A variable *completed* is set to 1 if the VM has completed its execution at the end of time $t-1$.

3.4.2 Admission Control

The pseudocode for the *AdmissionControl* module is presented in Algorithm 2. This module checks if there exists any RSU with sufficient storage to accommodate the data in the new VM. M_{new} denotes the list of new VMs that arrive at the beginning of time t and M_{old}^{data} denotes the list of VMs which are not new but have some new data generated till the end of time $t-1$. The algorithm first updates the VM's requirements based on the scheduling information of the last time step and data generation information (Lines 5-13). For all VMs, if a VM is scheduled in the last time step, then the processed data is subtracted from the current need. Similarly, if any new data is generated for the VM, it is added to its storage requirement. Then it checks if a new VM or an old VM with data can be assigned to any RSU based on the VM's storage requirement (Line 15). In case a suitable RSU is found, the VM is added to the system, otherwise, the VM is dropped temporarily. Dropped VMs are stored in a list *temp_dropped*. The VMs in *temp_dropped* are considered again for *AdmissionControl* in the next time step (Line 4).

Algorithm 2: AdmissionControl

```
1  M_t[j] ← Array of all existing VMs at time t
2  M_new[] ← new VMs at the beginning of time t
3  M_old^data[] ← Old VMs with some data generated till
   the end of time t − 1
4  Append temp_dropped to M_new
5  for (each VM i in M_t) do
6      if (M_{t−1}[i].scheduled) then
7          if (M_t[i].data) then
8              M_t[i].s − = D^r
9              M_t[i].s + = M_t[i].data_amount
10         M_t[i].s − = D^r
11     else
12         if (M_t[i].data) then
13             M_t[i].s + = M_t[i].data_amount
14 for (each VM i in M_New ∪ M_old^data) do
15     if (∃j : R_t[j].remaining_s ≥ storage need of VM i and Count_r! = 0) then
16         Add i to M_t
17         Count_r = Count_r - 1;
18     else
19         Add i to temp_dropped
```

3.4.3 VM Assignment

This module takes the admitted VMs and assigns the VMs in an iterative manner until all VMs are assigned or no assignment is possible. It then updates the status of VMs and RSUs.

Algorithm 3 shows the pseudo code for *VMAssignment*. In each iteration, it first chooses a subset of VMs to be considered in this iteration and then assigns them to RSUs using a matching-based algorithm. The status of the assigned VMs and the RSUs they are assigned to are then updated, and the next iteration is entered to assign the remaining VMs. This process continues until all VMs are assigned or no assignment can be done in an iteration.

The set of VMs to be considered is first updated by removing the set of VMs, M_{over}, which completed execution during time $t-1$. To choose a subset of VMs from all VMs (Line

Algorithm 3: VMAssignment

```
1  M_t ← M_{t-1} − M_over
2  VMAssigned = φ
3  flag = 1
4  while (M_t! = VMAssigned and flag == 1) do
5      choice = a subset of VMs from M_t chosen as per description and whose deadlines
       are nearer
6      Assigned = subset of choice assigned by the Hungarian Matching algorithm as
       per description
7      if (Assigned[] == φ) then
8          flag = 0
9          continue
10 VMAssigned = VMAssigned ∪ Assigned
11 Update M_t for VMs assigned
12 Update R_t for RSUs used in assignment
```

5), it places all the VMs present in an RSU into two different lists – the list of VMs which were scheduled at time $t-1$ and the list of VMs which were not scheduled at time $t-1$. The second list is given more priority than the first while choosing VMs to be considered for assignment. This is done because prioritizing the VMs which were not scheduled in the last time step increases their chance of being assigned to an RSU where they may be scheduled at time step t, which may in turn increase the number of VMs that complete their execution within their lifetime. The algorithm sorts all the VMs in ascending order of remaining lifetime so that the VMs whose lifetimes are nearer are given higher preference while selecting VMs for assignment. The chosen VMs are stored in a list *choice*, which is considered for assignment in the next time step.

The assignment of the chosen VMs to RSUs is done using the Hungarian Matching algorithm (Line 6). A bipartite graph is formed by considering a node for each VM in one partition and a node for each RSU in the other partition. Dummy RSUs (with zero resources) and VMs (with zero storage and computation requirements) are added as needed to make the number of nodes equal in both partitions. Edges are then added between all VMs and all RSUs to make the graph a complete bipartite graph as needed for running the Hungarian Matching algorithm. The weight of an edge between a VM i and an RSU j is chosen as follows. The weight is set to 0 if the remaining storage capacity j is less than the

storage requirement of i, otherwise,

$$weight = w_1(R_t[j].remaining_s - M_t[i].data\_remaining)/ \\ S\_MAX + \ w_2(R_t[j].remaining_c - M_t[i].c)/C\_MAX).$$

Here, S_MAX is the maximum of the maximum remaining storage of any RSU and maximum storage need of any VM, and C_MAX is the maximum of the maximum remaining computation of any RSU and maximum computation need of any VM. Thus, a VM is assigned to an RSU only if the RSU has enough storage to accommodate the VM. Also, preference is given to RSUs which have more storage and more computation capacity compared to the need of the VM. The importance of storage and computation controlled by two weights w_1 and w_2, $0 \leq w_1, w_2 \leq 1$ and $w_1 + w_2 = 1$. The Hungarian Matching algorithm is then run to find a maximum weighted matching on this graph. In this work, we have considered $w_1 = w_2 = 0.5$. Note that, the Hungarian Matching algorithm will assign each VM to an RSU as it finds a perfect matching; however, some VMs may be assigned to a dummy RSU. Such assignments are discarded and the corresponding VM is considered again in the next iteration.

3.4.4 Scheduling of VMs

Algorithm 4: VMScheduled

```
1 for each RSU i in R_t do
2     vm_edf ← sort R_t[i].v in earliest deadline first order;
3     for each VM j in vm_edf do
4         if ((!vm_edf[j].completed) and(R_t[i].remaining_c > vm_edf[l].c)) then
5             M_t[j].data_remaining -= D^r
6         else if (vm.edf[j].completed) then
7             Append vm.edf[j] to M_over;
8             vm.edf[j].r.remaining_c += vm.edf[j].c;
```

VMAssignment guarantees that the storage requirement of a VM will be met by the RSU it is assigned to. However, the RSU may not have sufficient computation capacity to schedule all the VMs assigned to it. The *VMScheduled* module chooses the VMs to be

scheduled (Algorithm 4). The choice of VMs is made simply on the basis of an *earliest-deadline-first* policy. In particular, for every RSU, it sorts the VMs assigned to the RSU in a non-decreasing order of the remaining lifetime. The longest prefix of this list of VMs whose total computation capacity does not exceed the computation capacity of the RSU is then scheduled for execution. The VMs not scheduled will not get executed in the current time step. The VMs scheduled for execution are assumed to be executed and their status is updated accordingly.

3.4.5 An Illustrative Example of the Proposed Approach

In this Subsection, we give an example to demonstrate the working of proposed algorithm. We have taken three RSUs (r_1 to r_3), five vehicles (v_1 to v_5), and two applications (a_1 and a_2) of same type (both generating data periodically). Table 3.2, Table 3.3, and Table 3.4 show the values chosen in the example. Table 3.5 shows the values for other parameters for this example. We are assuming that each vehicle requests for only one application at a time, as mentioned in the assumptions.

Table 3.2: *RSU capacities*

RSUs	r_1	r_2	r_3
Storage Capacity	60MB	60MB	60MB
Computation Capacity	10MHz	10MHz	10MHz

Table 3.3: *VM resource requirement*

Vehicles (VM created at RSU)	$v_1(VM_1)$	$v_2(VM_2)$	$v_3(VM_3)$	$v_4(VM_4)$	$v_5(VM_5)$
Storage Need	20MB	20MB	20MB	20MB	20MB
Computation Need	5MHz	5MHz	5MHz	5MHz	5MHz

Table 3.4: *Applications*

Applications	a_1	a_2
Deadline	8 time units	10 time units
Application Type	Periodic	Periodic

Table 3.5: *Network setting*

Data Generation Rate	2 MB/time unit
Data Processing Rate by RSU	10 MB/time unit
Data Transfer Rate	10 MB/time unit
Data Generation Range	4 time units
Number of Channels	1

At time $t = 0$, v_1, v_2, and v_3 are in the system. None of them are in the transmission range of any RSU. Although they have some initial data already present to be processed at $t = 0$, no request is there to any RSU as shown in Table 3.6. No VM is created at $t = 0$.

At $t = 1$, v_4 and v_5 join the network. Let v_1, v_2, v_3, and v_4 be in the transmission range of r_1 and v_5 be in the transmission range of r_2. Vehicles (v_1, v_2, v_3, and v_4) send their requests to r_1 (when they get to access the channel for transmission). RSU r_1 accepts the request of v_1 and r_2 accepts the request of v_5 (Admission Control Module). Corresponding VMs are created in r_1 and r_2. At the end of time step 1, v_1 has 14 MB ($22 - 10 + 2 = 14$ MB) of data. In this time step, 10 MB of data is transferred to RSU r_1 and 2 MB of data is generated by the application a_1 in vehicle v_1. The same procedure is followed for other vehicles as shown in Table 3.6. At each time instant, the remaining storage of an RSU is updated based on the amount of data transferred by an application of a vehicle.

At $t = 2$, the request of vehicle v_2 is accepted and admitted to r_1 (Admission Control). Then, on the basis of the proposed matching based algorithm, it is migrated to r_3 (VM Assignment Module). At $t = 3$, v_3's request gets accepted by r_1 and VM_3 is created. Vehicle v_1 has only 6 MB of data at the beginning of the time instant. Therefore, only 6 MB of data is transferred to RSU r_1 and 2 MB of data is generated at $t = 3$ which is processed at $t = 4$.

At $t = 4$, vehicle $v_4's$ request gets accepted by r_1. Thus, VM_4 gets created at r_2 (Admission Control Module). Then in the assignment phase, VM_4 is migrated from r_1 to r_2 (VM Assignment Module). Three VMs (VM_1, VM_2, and VM_5) complete their execution as they have no unprocessed data and no data is going to be generated (refer Table 3.5). VM_1, VM_2, and VM_5 scheduled in r_1, r_3, and r_2 respectively. This example shows the overall working of the proposed AALB algorithm.

3.4.6 Time Complexity Analysis

The time complexity of the proposed algorithm AALB can be computed from the time complexities of its modules, *AdmissionControl*, *VMAssignment*, and *VMScheduled*. The *AdmissionControl* module takes $O(X)$ time to complete, where X is the total number of vehicles. The *VMAssignment* module takes $O(X^2)$ time to complete. The *VMScheduled* module takes $O(XY)$ time to complete, where Y is the total number of RSUs. Thus, the algorithm AALB has the time complexity of $O(X^2 + XY)$ per time unit.

Table 3.6: *Example scenario*

Time (t)	**Vehicles Status (storage need, application) at the end of t**	**RSU Status (remaining storage, remaining computation, vehicles in the range) during time t**	**Requests (Vehicle to RSU) at the beginning of t**	**VMs in RSUs**	**Migration (From RSU→ To RSU) using matching-based technique**
0	$v_1(22, a_1)$ $v_2(22, a_2)$ $v_3(22, a_1)$	$r_1(60, 10, \{\})$ $r_2(60, 10, \{\})$ $r_3(60, 10, \{\})$	No request	No VM present in the system	No migration
1	$v_1(14, a_1)$ $v_2(24, a_2)$ $v_3(24, a_1)$ $v_4(22, a_2)$ $v_5(12, a_2)$	$r_1(50, 5, \{v_1, v_2, v_3, v_4\})$ $r_2(50, 5, \{v_5\})$ $r_3(60, 10, \{\})$	$v_1 \rightarrow r_1$ $v_2 \rightarrow r_1$ $v_3 \rightarrow r_1$ $v_4 \rightarrow r_1$ $v_5 \rightarrow r_2$	$r_1\{VM_1\}$ $r_2\{VM_5\}$	No migration
2	$v_1(6, a_1)$ $v_2(16, a_2)$ $v_3(26, a_1)$ $v_4(24, a_2)$ $v_5(4, a_2)$	$r_1(40, 5, \{v_1, v_2, v_3, v_4\})$ $r_2(40, 5, \{v_5\})$ $r_3(50, 5, \{\})$	$v_2 \rightarrow r_1$ $v_3 \rightarrow r_1$ $v_4 \rightarrow r_1$	**Admission Control:** $r_1\{VM_1, VM_2\}$, $r_2\{VM_5\}$ **VM Assignment**: $r_1\{VM_1\}$, $r_2\{VM_5\}$, $r_3\{VM_2\}$	VM_2 migrated from r_1 to r_3
3	$v_1(2,a_1)$ $v_2(8, a_2)$ $v_3(18, a_1)$ $v_4(26, a_2)$ $v_5(2,a_2)$	$r_1(24, 0, \{v_1, v_2, v_3, v_4\})$ $r_2(36, 5, \{v_5\})$ $r_3(40, 5, \{\})$	$v_4 \rightarrow r_1$ $v_3 \rightarrow r_1$	$r_1\{VM_1, VM_3\}$ $r_2\{VM_5\}$ $r_3\{VM_2\}$	No migration
4	$v_1(0,a_1)$ $v_2(0, a_2)$ $v_3(8, a_1)$ $v_4(16, a_2)$ $v_5(0,a_2)$(No data generation in this time step)	$r_1(12, 0, \{v_1, v_2, v_3, v_4\})$ $r_2(24, 0, \{v_5\})$ $r_3(32, 5, \{\})$	$v_4 \rightarrow r_1$	**Admission Control:** $r_1\{VM_1, VM_3, VM_4\}$ $r_2\{VM_5\}$ $r_3\{VM_2\}$ **VM Assignment:** $r_1\{VM_1, VM_3\}$ $r_2\{VM_5, VM_4\}$ $r_3\{VM_2\}$	VM_4 migrated from r_1 to r_2, VM_1, VM_2 and VM_5 completes their execution

3.5 Results and Discussion

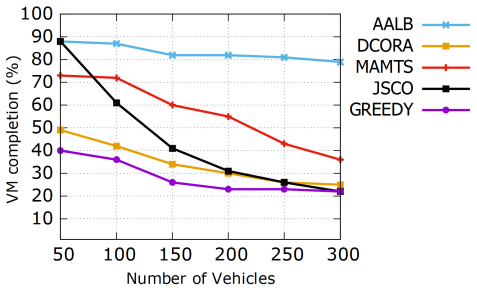
The proposed algorithm AALB is simulated using a Java-based discrete event simulator developed for the implementation of thesis work. While designing the simulator the network parameters are taken into consideration. To check the correctness of the simulator, the performance is checked with a boundary condition. The results are not violating the lower bound condition. We have also done a delay analysis in Chapter 4.5 to validate our simulation results, as shown in Figure 4.8 of the thesis. Our simulation consists of a city scenario with bidirectional roads and an area of 10 km^2. Vehicular traffic is generated using Simulation in Urban Mobility (SUMO) [18] by considering the lower Manhattan city scenario. The simulation parameters are given in Table 3.7. Major parameters for simulation are taken from the works [22, 27].

Table 3.7: *Simulation parameters for AALB*

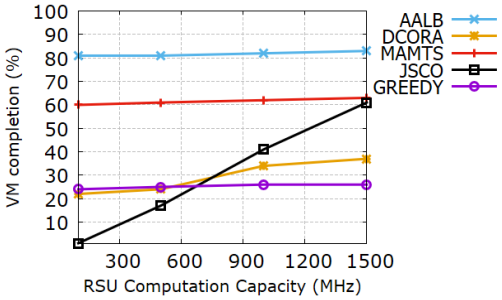
Parameters	**Values**
Number of time steps	890
Number of RSUs	9
Data processing per unit time at RSU	10 Mbps
Transmission range of RSU	500 meters
Storage capacity of RSU	6000 MB
Computing capacity of RSU	1000 MHz
Range of VM's initial storage need	100 MB - 300 MB
Range of VM's computing need	5 MHz - 40 MHz
Fixed migration cost of VM	0.010 $
Migration cost per unit of data transfer	0.002 $
Data generation rate of periodic application	10 Mbps
Amount of data generated per event	500 MB
The inter arrival time for event-driven task	100 Sec
Distribution used for event generation	Exponential
Number of channels	4
Area in the city considered for simulation	10 km^2

We have evaluated our proposed algorithm with different types of applications having variable data generation rates. We have considered periodic applications, event-driven applications, and their combination for simulation. Periodic applications generate data peri-

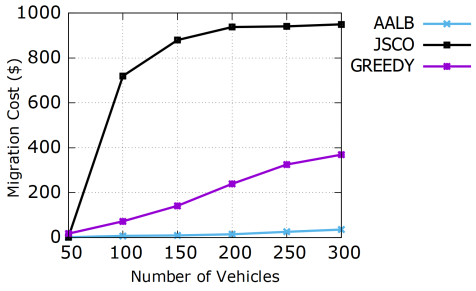
odically and event-driven applications generate data based on the occurrence of events, their combination consists of both periodically generated data and event-driven data. Further, we have compared AALB with three existing algorithms in the literature – DCORA [31], MAMTS [32], and JSCO [30]. Along with these algorithms, we have also compared AALB with a baseline algorithm where the VMs are only migrated to the RSUs in the vehicle's route. We call the baseline algorithm as GREEDY algorithm. The amount of data generated per event in event-driven applications is dependent on the type of applications. We have experimented with two different values, 500 MB and 250 MB based on the amount of data generated in different applications [50].



(a) *Variation of VM completion percentage with number of vehicles*



(b) *Variation of VM completion percentage with computation capacities of RSU*



(c) *Variation of VM migration cost with number of vehicles*

Figure 3.4: *Performance for periodic applications*

3.5.1 Periodic Applications

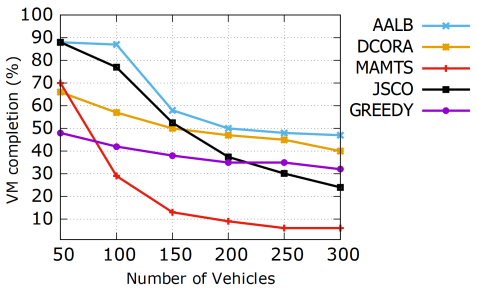
In this Subsection periodic applications are considered for simulations where applications generate data periodically with a data generation rate of 10 Mbps.

Figure 3.4a shows the variation of VM completion percentage with the number of vehicles. It can be observed that the percentage of VM completion is higher in the case of AALB as compared to DCORA, MAMTS, JSCO, and GREEDY algorithms with an increase in number of vehicles. This is because of the efficient VM migration technique used in AALB when an RSU is overloaded. Figure 3.4b shows the variation of VM completion percentage with computation capacities of RSUs. It can be seen that AALB shows better performance than the other four algorithms. This is because of the fact that AALB prioritizes VMs which were not scheduled in the last time steps. The trends for each algorithm show a linear pattern except JSCO. This is because, JSCO only focuses on the computation resource while the other four algorithms are dependent on other factors apart from computation. AALB and GREEDY depend on both storage and computation, DCORA depends on utilization, and MAMTS depends on other applications. Therefore, the effect of the increase in computation capacity is clearly visible in the JSCO output. Figure 3.4c shows the variation of VM migration cost of JSCO, GREEDY, and AALB algorithm with number of vehicles. It can be observed that AALB shows lower costs than JSCO and GREEDY algorithms. This is because of the efficient VM assignment technique used in AALB. We have compared VM migration cost of AALB with JSCO and GREEDY algorithms. This is because, DCORA and MAMTS algorithms do not focus on the migration cost. The migration cost of JSCO is more as compared to AALB and GREEDY algorithms. This is because, JSCO focuses only on the percentage of VM completion without considering VM migration cost.

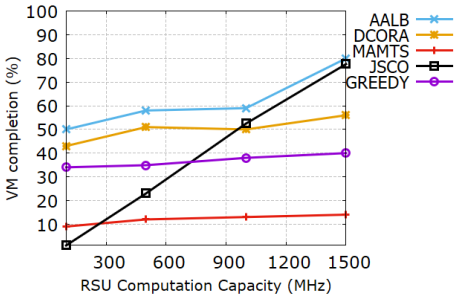
3.5.2 Event-Driven Applications

In this Subsection event-driven applications are considered for simulations where applications generate data when an event occurs. In-vehicle infotainment and anomaly detection from the video are examples of applications that generate data of nearly 500MB per second. Several other applications, such as traffic management applications, emergency braking systems, online gaming, etc. are also considered in this work [49]. The task considered in this work is either periodic or event-driven task. Traffic management, online gaming are examples of periodic applications where anomaly detection and emergency breaking systems are even-driven tasks. Figure 3.5 shows the results for the algorithms when 500 MB data is generated per event. Figure 3.6 shows the results when 250 MB data is generated per event. The generation of events follows exponential distribution in both cases.

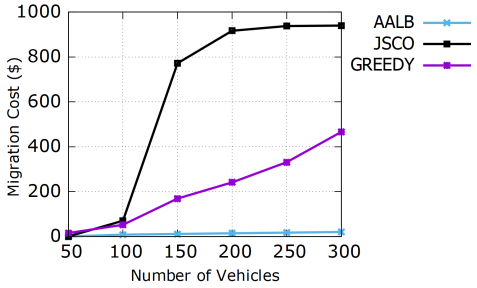
Figure 3.5a shows the variation of VM completion percentage with the number of vehicles in the case of event-driven applications. The performance of AALB is better in comparison to the other four algorithms. The drop in overall VM completion percentage is more in event-driven applications as compared to periodic applications. This is because, the data generated per event is larger in event-driven applications. The amount of data generated increases as time proceeds. This leads to more load and less time to complete the task. Therefore, VM completion percentage is less as compared to that of periodic applications.



(a) *Variation of VM completion percentage with number of vehicles*



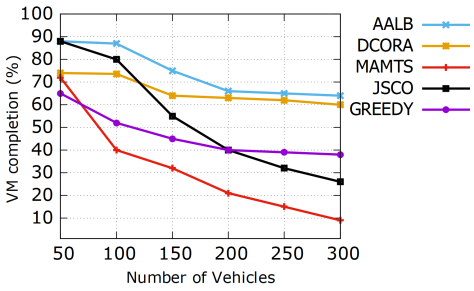
(b) *Variation of VM completion percentage with computation capacities of RSU*



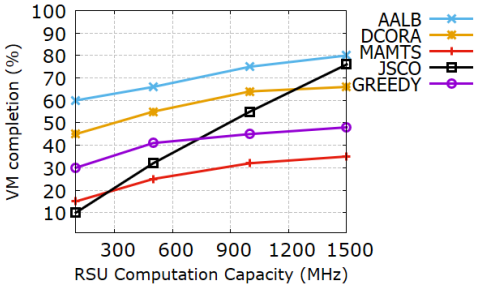
(c) *Variation of VM migration cost with number of vehicles*

Figure 3.5: *Performance for event-driven applications (500MB/event)*

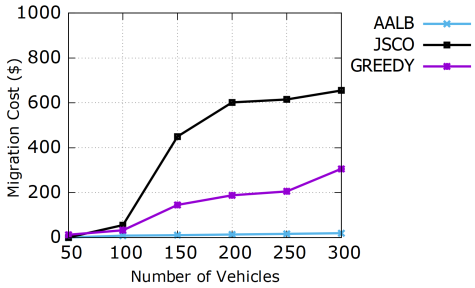
Figure 3.5b shows the performance of all algorithms with an increase in computation capacities of RSUs. It can be observed that the VM completion percentage of AALB shows better performance while varying the computation capacities of RSUs. The reason behind such performance is the use of efficient matching-based techniques for VM assignment and consideration of status of VMs in the last time step. In Figure 3.5c, output of VM migration cost is shown with varying the number of vehicles for event-driven applications. It can be observed that the performance of AALB for migration cost is better in comparison to JSCO and GREEDY algorithms. The reason is the same as mentioned in subsection 3.5.1.



(a) *Variation of VM completion percentage with number of vehicles*



(b) *Variation of VM completion percentage with computation capacities of RSU*



(c) *Variation of VM migration cost with number of vehicles*

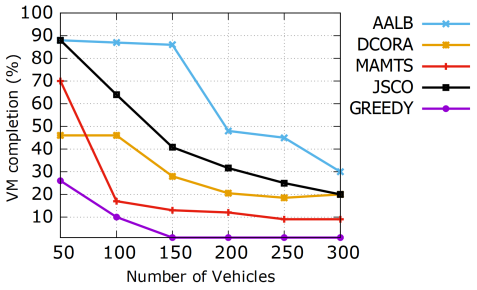
Figure 3.6: *Performance for event-driven applications (250MB/event)*

Figure 3.6 shows the same results for event-driven applications where the amount of data generated per event is 250 MB. It can be observed in Figure 3.6a that the VM completion percentage is higher as compared to Figure 3.5a. This is because, the amount of data generated is less in this case. The VM completion percentage in Figure 3.6b shows a similar pattern as Figure 3.5b, the only difference being in the percentage of VM completion. Here, the percentage of VM completion is higher. This is again because of the lower amount of data generated per event. Figure 3.6c shows that AALB performs better than JSCO and GREEDY. This is because of the efficient VM allocation approach used in AALB. The reason behind comparing AALB with JSCO and GREEDY while ignoring MAMTS and DCORA is given in subsection 3.5.1.

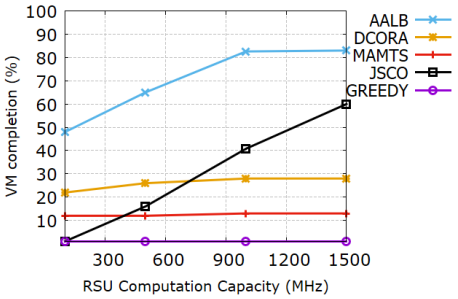
3.5.3 Periodic and Event Driven Applications

Figure 3.7 shows performance of five algorithms while considering both periodic and event-driven applications, where event-driven applications generate 500 MB of data per event.

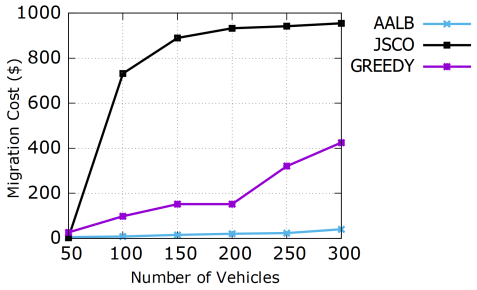
In Figure 3.7a, it can be observed that the percentage of VM completion is less as



(a) *Variation of VM completion percentage with number of vehicles*



(b) *Variation of VM completion percentage with computation capacities of RSU*

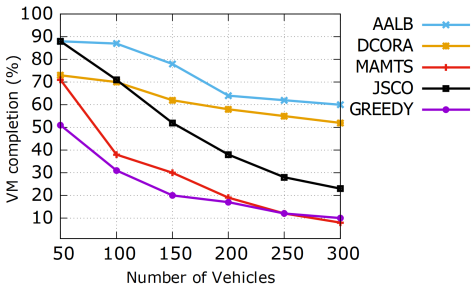


(c) *Variation of VM migration cost with number of vehicles*

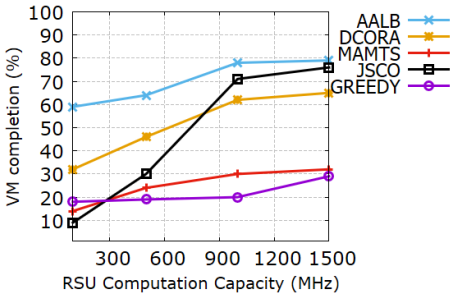
Figure 3.7: *Performance for periodic and event-driven applications (500MB/event)*

compared to either periodic or event-driven applications. The trends show a sharp drop in the VM completion percentage for all the algorithms. This is because, the amount of data generated in this case is more as compared to either an event-driven scenario or a periodic scenario. In this case also, AALB performs better in comparison to the other algorithms. This is because of the efficient VM assignment approach.

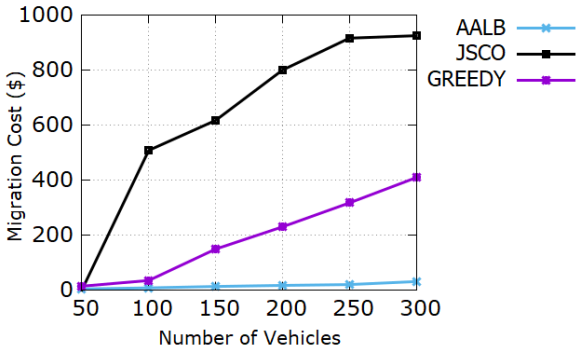
Figure 3.7b shows the variation of VM completion percentage with the computation capacities of RSUs. Here too, AALB shows improved performance compared to the other algorithms. This is because, AALB is independent of the time at which data is generated and uses an efficient VM allocation approach. In Figure 3.7b, AALB saturates after the computation capacity reaches 1000 MHz. This is because, AALB considers storage resources while doing VM assignments along with computation resources. Even though computation capacities of RSU increase, some VMs do not get sufficient storage resources in RSUs. VM migration cost of AALB, JSCO and GREEDY algorithms are compared in Figure 3.7c. In Figure 3.7c, it can be observed that the performance of AALB for migration cost is better as compared to JSCO and GREEDY algorithms. This is because, AALB assigns the VMs



(a) *Variation of VM completion percentage with number of vehicles*



(b) *Variation of VM completion percentage with computation capacities of RSU*



(c) *Variation of VM migration cost with number of vehicles*

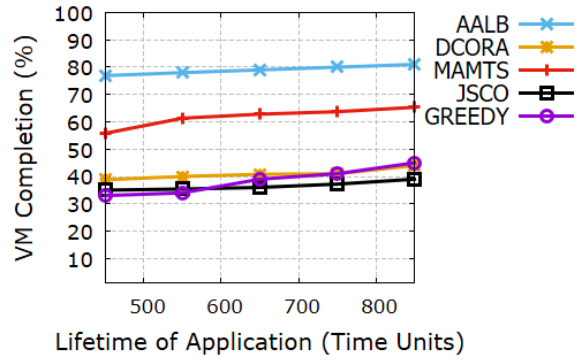
Figure 3.8: *Performance for periodic and event-driven applications (250MB/event)*

efficiently and minimizes the migration cost.

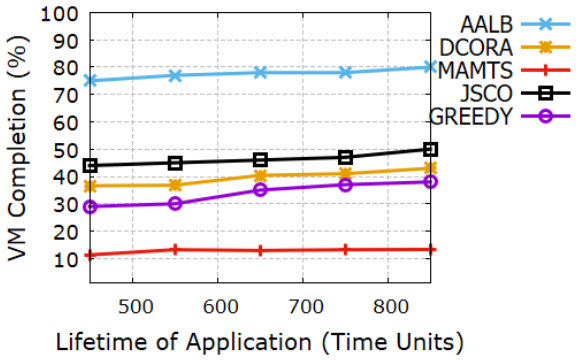
Figure 3.8 shows the results of all five algorithms, where data is generated by both periodic applications and event-driven applications. The amount of data generated per event in this case is 250MB. It can be seen in Figure 3.8a that the drop in VM completion percentage is not sharp. This is because of less data generation. Here also AALB works better than the other algorithms because of the same reason as discussed in subsection 3.5.1. In Figure 3.8c, AALB shows improved performance compared to the other algorithms because of efficient VM assignment approach. Figure 3.8c shows that AALB migration cost is less as compared to JSCO and GREEDY algorithms. The reason is the same as discussed in subsection 3.5.1.

3.5.4 Effect of Application Lifetime

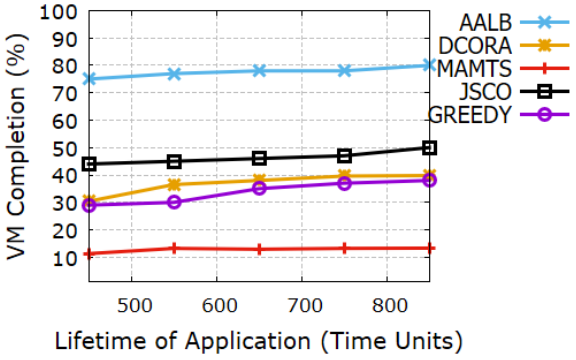
Here, we study the effect of application lifetime on the VM completion percentage. The lifetime of an application which is defined as the time within which the application's execution must be completed, is taken as the application delivery constraint in this work. The application lifetime is varied from 450 to 850 time units. Note that in our simulations, vehicles generate data to a maximum of 449 time units, and thus the range of application lifetime chosen represents applications with very tight delivery constraints to those with more loose constraints where more time is available to process the generated data.



(a) *Variation of VM completion percentage with lifetime of periodic applications*



(b) *Variation of VM completion percentage with lifetime of event-driven applications*



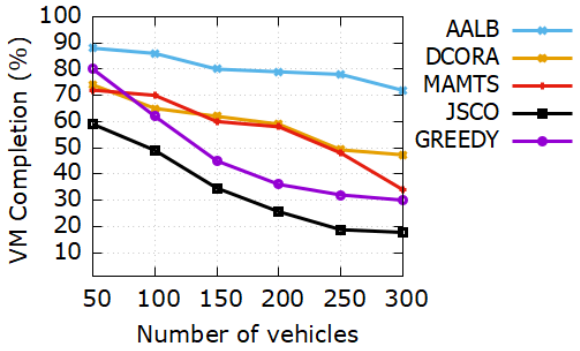
(c) *Variation of VM completion percentage with lifetime of periodic and event-driven applications*

Figure 3.9: *Performance of periodic, event-driven applications, and their combination with application lifetime*

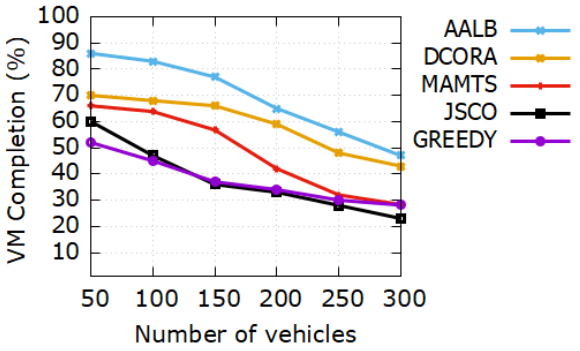
Figures 3.9a, 3.9b, and 3.9c show the variation in VM completion percentage with varying lifetimes of periodic applications, event-driven applications, and their combination respec-

tively. In all the three graphs, AALB shows improved performance as compared to the other algorithms for all lifetime values. This is because, AALB prioritizes earlier deadlines while scheduling VMs, and hence more VMs are able to finish within their lifetimes.

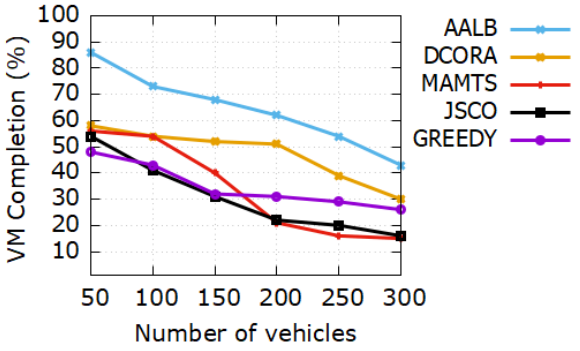
3.5.5 Effect of Multiple Applications in Vehicles



(a) *Variation of VM completion percentage with number of vehicles of periodic applications*



(b) *Variation of VM completion percentage with number of vehicles of event-driven applications*



(c) *Variation of VM completion percentage with number of vehicles of periodic and event-driven applications*

Figure 3.10: *Performance of periodic, event-driven applications, and their combination with varying numbers of vehicles running multiple applications*

A vehicle may run more than one application simultaneously. To check the effect of multiple applications running simultaneously on a vehicle, we have simulated a scenario in which two vehicles, each running a different application, move as a convoy and reach all RSUs in the route simultaneously. This scenario essentially simulates a single vehicle running two applications simultaneously.

Figures 3.10a, 3.10b, and 3.10c show the variation of VM completion percentage with number of vehicles for periodic applications, event-driven applications, and a combination of periodic & event-driven applications respectively. It can be observed that for all three cases, AALB shows better results as compared to other algorithms because even with an increase in the number of applications per vehicle, the corresponding VMs will be executed independently at different RSUs using the efficient method of matching the VMs with RSUs. The applications considered in this scenarios are independent of each other.

3.5.6 Effect of Contention at RSUs

Contention at any RSU for data transfer from vehicles is dependent on the number of vehicles whose contact times at the RSU overlap. To study the effect of RSU contention on our algorithm, we vary the time range within which all vehicle requests are generated in our simulation. A lower value of the time range implies more vehicles will arrive at an RSU around the same time causing more contention, and vice-versa. The number of vehicles considered for this simulation is 300, so as to achieve a high density of vehicles that arrive more or less at the same time at RSUs.

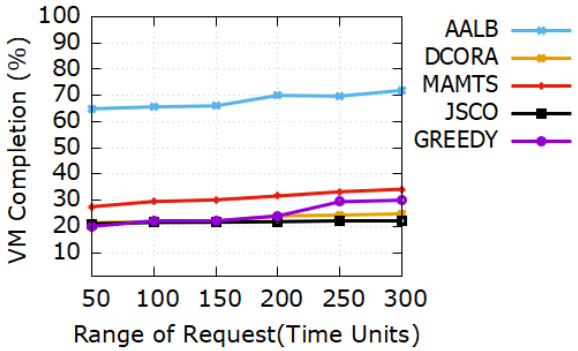
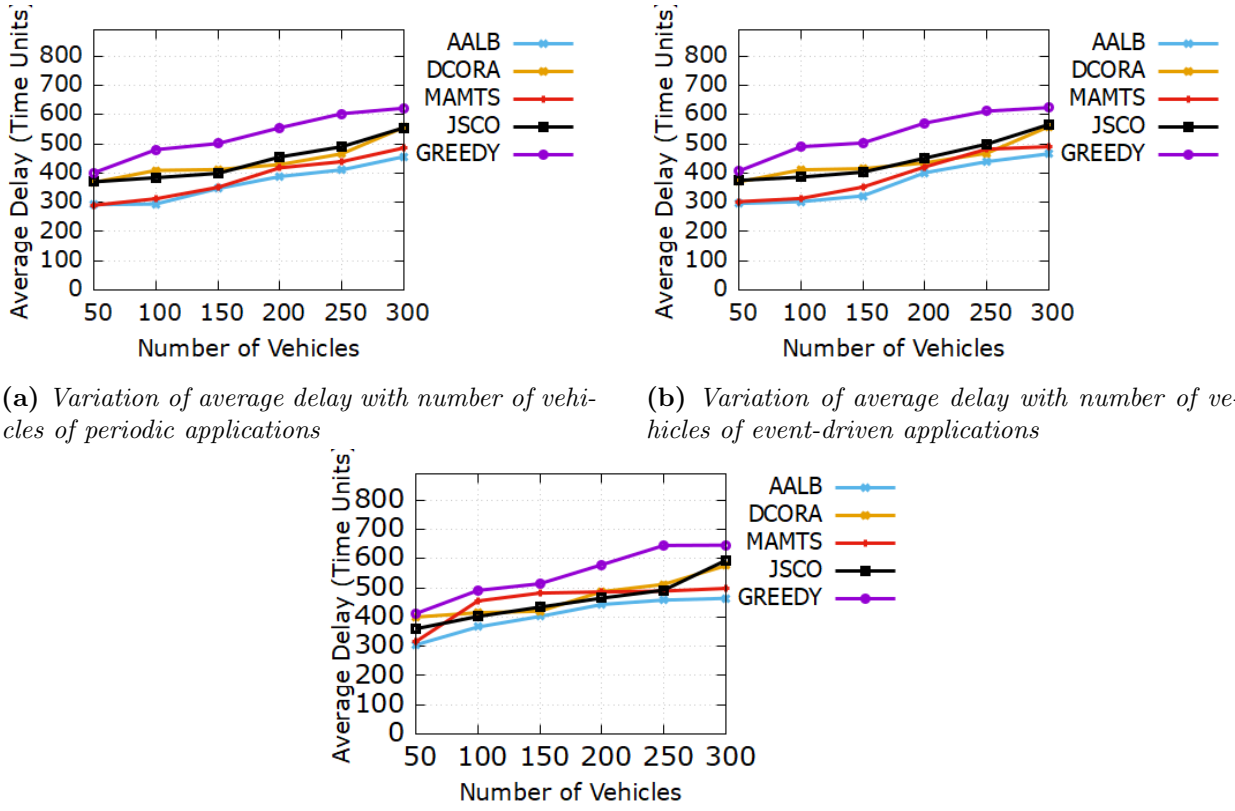


Figure 3.11: *Variation of VM completion percentage with time range of requests*

Figure 3.11 shows the variation of VM completion percentage with the increase in the contention at RSUs. The x-axis represents the time range of generating the requests in simulation time units. It can be observed that AALB shows improved performance as compared to other algorithms. This is primarily because contention is avoided by AALB by admitting the requests to any RSUs in the network if sufficient storage is available. Thus, it decreases the waiting time for the number of requests. This leads to better performance of AALB over other algorithms.

3.5.7 Average Delay

We next study the average delay incurred by the requests generated by the vehicles. We define delay as the difference in the time between the completion of a VM and the time of generation of a VM.



(a) *Variation of average delay with number of vehicles of periodic applications*

(b) *Variation of average delay with number of vehicles of event-driven applications*

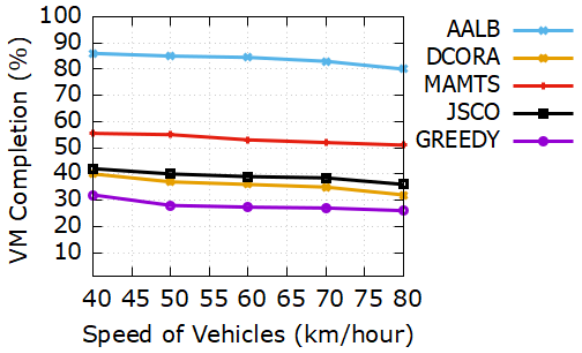
(c) *Variation of average delay with number of vehicles of periodic and event-driven applications*

Figure 3.12: *Average delay for periodic applications, event-driven applications, and their combination*

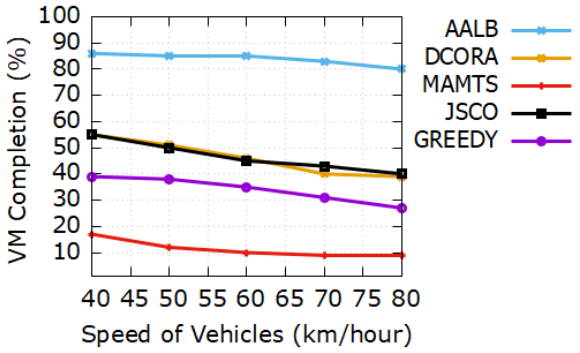
Figures 3.12a, 3.12b, and 3.12c show the variation of average delay with number of vehicles for periodic applications, event-driven applications, and a combination of periodic and event-driven applications, respectively. It can be observed that the performance of AALB is better as compared to the other existing algorithms in all cases. This is because, in general, when an RSU gets overloaded, it is unable to accept any further requests from its neighboring vehicles, leading to an increase in delay incurred by the request. However, in AALB, the requests are forwarded to any of the RSUs in the network which has enough resources

available to accept the request, thereby giving an improved performance as compared to all the other algorithms.

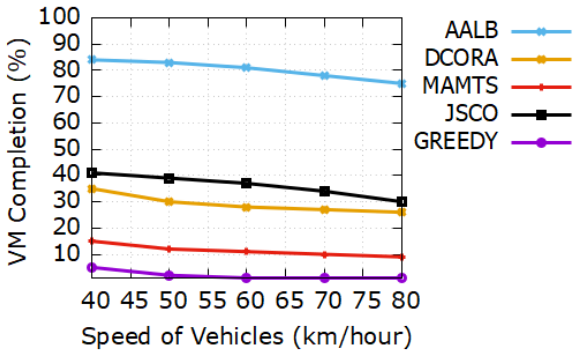
3.5.8 Effect of Vehicle Speed



(a) *Variation of VM completion percentage with the speed of vehicles for periodic applications*



(b) *Variation of VM completion percentage with the speed of vehicles for event-driven applications*



(c) *Variation of VM completion percentage with the speed of vehicles for periodic and event-driven applications*

Figure 3.13: *Performance of periodic applications, event-driven applications, and their combination with varying speed of vehicles*

We have studied the effect of speed variation of vehicles on the VM completion percentage of the system. We have considered the number of vehicles as 150 for this simulation and observed that an increase in speed of the vehicles leads to a decrease in the VM completion percentage. However, the effect is very less as shown in Figure 3.13. This is because the speed of the vehicles only affects the performance of the system till the time an application data is transferred to RSUs. Figure 3.13a shows the performance of the periodic application

with variation of speed. Figure 3.13b shows the performance of the event-driven application and Figure 3.13c shows the performance of the periodic & event-driven applications together with variation of speed.

It can be observed that AALB outperforms the three existing algorithms and the GREEDY algorithm. It is because of the efficient VM migration technique and wired medium chosen to transfer the data between RSUs.

3.6 Chapter Summary

This chapter discussed the approach to deal with load balancing considering application types and resource requirements of each application request. Periodic applications, event-driven applications, and their combination are considered for evaluation. The proposed algorithm AALB is compared with existing algorithms. Effect of vehicle speed, multiple applications are studied along with the available resources of RSUs. In case of periodic applications, the VM completion percentage of AALB improves by 36.67% over MAMTS, 100% over JSCO and by 134.29% over DCORA.

4

Resource Renting for Load Balancing in Vehicular Networks

This chapter presents our second contribution. Here, we have proposed one pricing model which is applied to the RSUs and their resources while doing the load balancing. The requests generated by the vehicles are assigned to one of the RSUs in the networks. Each RSU is considered as one independent entity which charges a fixed cost for sharing its resources with any other RSUs in the network.

4.1 Introduction

Renting resources is a widely used idea in cloud computing environments where the resources of one entity are rented based on one price scheme. It increases the service availability in case of resource constraints environment. It reduces the cost of providing the services to end users without deployment of vast network infrastructure. The price offered by the service providers are generally two types -1) Fixed price and 2) Dynamic price [57]. In the case of a fixed price, the price per unit of resource per unit of time does not vary. However, in the case of dynamic price, the unit cost for a unit of resource is dependent on the availability of the resources. Price per unit resources increases with a decrease in available resources. In our work, we have focused on fixed pricing to reduce the time for decision making.

4.2 Problem Formulation

We have formulated the issues raised by this study as an optimization problem in this Section. We define the input and output variables. The afterword constraints and objectives are specified with a detailed explanation. Our goal is to maximize the number of requests processed and minimize the delay and total cost. The total cost is a combination of rent-out cost and migration cost.

4.2.1 Input Variables

Let T be the total amount of time considered for the simulation. Total X number of vehicles are there which is represented by set $V = \{v_1, v_2, \ldots, v_X\}$ and Y number of RSUs are there which is represented by set $R = \{r_1, r_2, \ldots, r_Y\}$. Vehicle $v_j \in V$, is expressed by a tuple $\langle P_j, a_j, \mathcal{L}_j, \wedge_j \rangle$, where

- Route of v_j is specified as a series of RSUs $\langle r_1^j, r_2^j, \ldots, r_l^j \rangle$ and is called P_j, where l is a variable which is dependent on the total number of RSUs in the path of v_j
- The vehicle's application is called a_j
- The application deadline is $\mathcal{L}_j$ and it runs at v_j
- Application's data generation rate for v_j is $\wedge_j$.

Generated data $\wedge_j$ is represented by $\langle \lambda_1^j \ \lambda_2^j, \ldots \lambda_T^j \rangle$ as a sequence, where the quantity of data produced by the vehicle v_j at time slot k is $\lambda_k^j \geq 0$. Additionally, the arrival and departure times of each RSU r_i in the route of v_j are shown as $arr_{r_i}^j$ and $dep_{r_i}^j$, respectively. RSU r_i is represented by the tuple $\langle s_i, p_i, D_i^r \rangle$, where, s_i is the storage capacity, p_i is the computing capacity, and D_i^r is the DPR of RSU r_i.

A set of applications is represented by the notation $A = \{a_1, a_2, \ldots, a_k\}$. Each application a_k is represented by a tuple $\langle s_k, p_k, Cost^f \rangle$, where s_k denotes the amount of storage required for application a_k, p_k denotes the amount of computation required for application a_k, and $Cost^f$ denotes the fixed cost of migration.

Let $B = \{b_1, b_2, \ldots b_X\}$ be the set of VMs generated for X vehicles in some RSU $r_i \in R$. Let Q_i^t represent rent-out cost for the resources offered by RSU r_i at time t and M_i^t is the migration cost of VM from RSU i.

4.2.2 Output Variables

Output variables are expressed by two sets $\{x_{ijt}\}$ and $\{y_{ijt}\}$, $\forall v_j \in V$, $\forall r_i \in R$, and $\forall t, 1 \leq t \leq T$. In the case of VM b_j,

- If b_j is in RSU r_i at time t, x_{ijt} is set to 1; otherwise, it is set to 0
- If b_j is scheduled in RSU r_i at time t, y_{ijt} is set to 1; otherwise, it is set to 0.

Difference between data sent by vehicle v_j to RSU until time t and the data processed by RSUs until time t for vehicle v_j can be used to compute the quantity of unprocessed data for VM b_j at time instant t. This is represented as,

$$U_j^t = \sum_{k=1}^{arr_{r_c}^j} \lambda_k^j - D_i^r \sum_{i=1}^{Y} \sum_{t=1}^{T} y_{ijt} \tag{4.1}$$

Here, r_c is RSU in v_j's path just before time instant t.

An indicator variable z_j^{tki} is defined, where its 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. z_j^{tki} indicates that the VM b_j has migrated from one RSU r_k to other RSU r_i at time t. Therefore,

$$z_j^{tki} = \begin{cases} 1, & \text{if VM } b_j \text{ migrated from RSU } k \text{ to } i \text{ at time } t \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

if $k = i$, VM b_j is not moved to another RSU, and there is no migration cost. We also define an indicator variable I_j, such that,

$$I_j = \begin{cases} 1, & \text{if } U_j^t = 0, \text{ if } v_j \text{ departs from last RSU on its route} \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

4.2.3 End-to-End Delay

Delay is a crucial parameter in VANETs due to the dynamic nature of network topology. In VANETs, vehicles often stay within the coverage area of an RSU for a brief period due to mobility, resulting in incomplete data transfers and increased waiting times for VMs at RSUs. We defined end-to-end delay as the duration of processing a request, starting from the request generation to the completion of data processing. It includes factors, such as data transfer delay, waiting delay of the VMs at RSUs, VM migration delay between RSUs, and the execution delay of the VM within RSUs.

Let D_j be the delay incurred by VM created for vehicle v_j. Let t_j^s be the time at which VM b_j gets created and t_j^f be the time at which VM b_j completes its processing. Delay is calculated as follows:

$$D_j = t_j^f - t_j^s \tag{4.4}$$

4.2.4 Objectives

In this work, our objectives are maximizing the number of completed VM executions, minimizing the delay incurred, and reducing the overall cost associated with VM migration. By addressing these objectives, we aim to improve the efficiency and performance of RSUs in processing the substantial amount of data generated by vehicles. Through our proposed algorithm, Efficient Resource Renting (ERR), we maintain a balance between resource allocation, cost management, and timely execution of VMs. Hence, we can define the objectives as,

$$maximize \sum_{j=1}^{X} I_j \tag{4.5}$$

$$minimize \sum_{j=1}^{X} \sum_{t=1}^{T} \Big((z_j^{tki})(Q_i^t + M_i^t) \ + \ (1 - z_j^{tki})(Q_k^t) \Big) \tag{4.6}$$

$$minimize \left(\sum_{j=1}^{X} E(D_j) \right) /(X) \tag{4.7}$$

Our goal is to maximize the number of requests processed as represented by Equation 4.5. Equation 4.6 is to reduce the overall expense of assigning, scheduling, and migrating VMs in the system. This objective considers cases when a VM migrates to some other RSUs and when there is no VM migration at the current time step. Equation 4.7 represents the objective to minimize end-to-end delay, where $E(D_j)$ is the expected delay for VM b_j.

4.2.5 Constraints

Following are the constraints in our problem scenario with the objectives specified above:

- After reaching the final RSU (r_l^j) on its route, a vehicle should not continue to generate data.

$$\lambda_t^j = 0, \quad \forall v_j \in V, \forall t > arr_{r_l}^j \tag{4.8}$$

- Until a vehicle reaches the first RSU in its path, no VM will be created for it.

$$x_{ijt} = 0, \quad \forall v_j \in V, \forall t, 1 \leq t < arr^j_{r_1} \tag{4.9}$$

- Between its arrival at the initial RSU and departure from the last RSU on its path, a VM should always be present in only one RSU at any given time t.

$$\sum_{i=1}^{Y} x_{ijt} = 1, \quad \forall v_j \in V, \forall t, arr^j_{r_1} \leq t \leq dep^j_{r_l} \tag{4.10}$$

- Only when a VM has some unprocessed data, then it can get scheduled.

$$y_{ijt} = 1 \Longrightarrow (x_{ijt} = 1 \wedge U^t_j \neq 0) \tag{4.11}$$

- An RSU's overall computing capacity should not be exceeded by the combined total computing needs of all of its VMs.

$$p_k \times \sum_{j=1}^{X} y_{ijt} \leq p_i, \quad \forall r_i \in R, \forall v_j \in V, \ \ \forall t, 1 \leq t \leq T \tag{4.12}$$

- An RSU's storage capacity should not be exceeded by the combined total storage requirements of all VMs.

$$(\sum_{j=1}^{X} U^t_j + \sum_{j=1}^{X} s^j_k) x_{ijt} \leq s_i, \ \ \forall r_i \in R, \forall v_j \in V, \forall t, 1 \leq t \leq T, \tag{4.13}$$

- A VM's total execution time should be less than the time at which the associated vehicle departs from the final RSU (r^j_l) on its route.

$$(\sum_{t=1}^{arr^j_{r_l}} \lambda^j_t)/(D^r_i \sum_{i=1}^{Y} \sum_{t=1}^{dep^j_{r_l}} y_{ijt}) \leq dep^j_{r_l}, \quad \forall v_j \in V \tag{4.14}$$

This problem is a mixed integer programming problem, which is NP-hard [58,59]. Therefore, we propose a heuristic method as a solution to this problem. We call the algorithm as Efficient Resource Renting (ERR).

4.3 Proposed Methodology

In this Section, we have proposed an Efficient Resource Renting (ERR) algorithm, which aims to maximize the number of VMs that complete execution by reducing the rent-out cost, total cost, and end-to-end delay. Total cost is the sum of rent-out cost and migration cost. While performing LB as described in Equation (4.19), ERR takes RSU's remaining storage, remaining computation, data processing rate, and rent-out cost into account. Rent-out cost is incurred when RSU's computational and storage resources are used. The problem scenario is represented by graphical methods. We propose a static pricing model which is used in ERR.

4.3.1 Pricing Model

We have proposed a pricing model for the resources of RSUs, where each RSU is associated with a per unit storage cost (S_i), and per unit computing cost (C_i) per unit of time. Additionally, we assume a predefined base cost per unit time (P_i) when a VM is present at an RSU but waiting for corresponding data to arrive. Using these definitions, we can calculate Q_i^t, which represents the rent-out cost of the RSU r_i at the current time step, based on the VM's storage and computing requirements. Now, let's explore the different scenarios for the cost:

- **Case 1:** *A VM is currently waiting for data to proceed with its execution.*

  Cost incurred at time step t is the base cost of VM b_j for staying at RSU r_i, and can be written as:

$$Q_i^t = P_i \tag{4.15}$$

- **Case 2:** *In the event that VM b_j, which is present in RSU r_i at time step t, has some data to process but has not yet been scheduled. Or, when a vehicle requests the formation of a VM, the RSU closest to the vehicle does not have enough resources, therefore, the VM is instead created in another RSU.*

In this instance, the cost of storage for the VM's unprocessed data is taken into consideration, and the total cost offered is:

$$Q_i^t = U_j^t S_i \tag{4.16}$$

where, the unprocessed data in VM b_j at time t is represented by U_j^t

- **Case 3:** *VM b_j has some amount of data to process and is scheduled to execute in one RSU.*

  In this case, for the VM's unprocessed data, we take into account storage as well as computing costs. This is specified as:

$$Q_i^t = U_j^t S_i + V_j C_i \tag{4.17}$$

  where, V_j is computing requirement of the VM b_j.

We additionally take into account the following costs if a migration of VM b_j to RSU r_i occurs at time t:

$$M_j^t = Cost_j^f + U_j^t * Cost_j^v \tag{4.18}$$

where, the fixed cost of VM migration is $Cost_j^f$. The migration cost for each unit of data transfer is $Cost_j^v$, and the migration cost for the VM b_j is M_j^t. The migration cost of the VM and the rent-out cost together make up the total cost.

4.3.2 Graphical Representation

Assignment of VMs to RSUs at time step t is depicted by a bipartite graph (Figure 4.1) consisting of X number of VMs and Y number of RSUs. Each VM that is still in execution at time t is represented by a node in one partition, while each RSU is represented by a node in the other partition. An edge between a VM node and an RSU node signifies that the corresponding VM can be allocated to that particular RSU.

The problem scenario considered here is a large-scale assignment problem. It can be solved by using the Hungarian algorithm which works well for balanced problems. This means, number of nodes in both sets of bipartite graphs are the same. To maintain equal

nodes in both partitions, dummy nodes (RSUs and VMs) are introduced as needed. The objective of the allocation process is to maximize the number of VMs completing their execution, while simultaneously minimizing overall cost and reducing end-to-end delay. The primary goal is to assign each VM to one RSU from the RSU set at time t. Finally, a weight is assigned to each edge as follows:

$$W_{ji} = \omega_1(R_t[i].s - (F_j + U_j^t)) + \omega_2(D_i^r) + \omega_3(-R_t[i].p/V[j].p) + \omega_4(-R_t[i].Q_i^t) \quad (4.19)$$

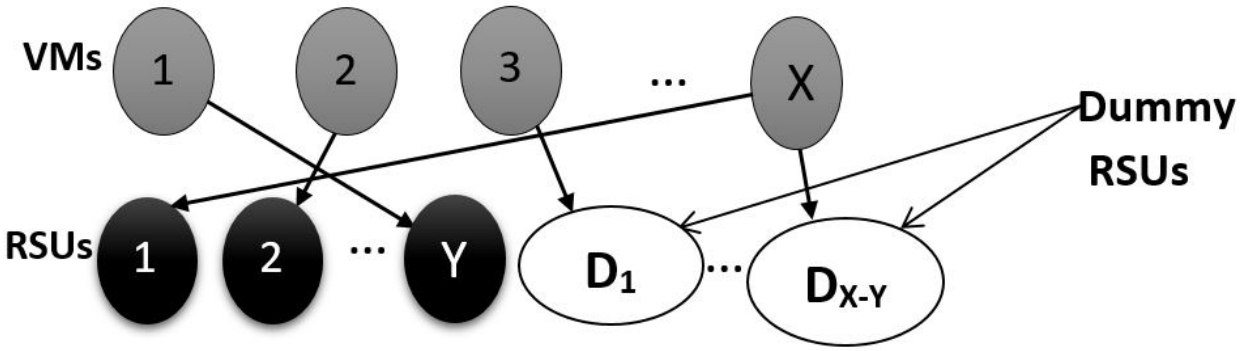


Figure 4.1: *Graphical representation*

Here, $\omega_1 + \omega_2 + \omega_3 + \omega_4 = 1$ and $0 \leq \omega_1, \omega_2, \omega_3, \omega_4 \leq 1$. The weights ω_1, ω_2, ω_3, and ω_4, respectively, determine the relative importance of storage resources, DPR, compute resources, and rent-out cost. Note that, these weights can be adjusted depending on the type of applications and their requirements. The weight of the edge at time t between nodes r_i and v_j is W_{ji}. $R_t[i].s$ is the remaining storage available with RSU r_i. F_j is the data to be generated till the deadline of the application j. D_i^r is the DPR of RSU r_i. U_j^t is the amount of data yet to be processed in VM b_j. $R_t[i].p$ is the amount of remaining computing resources of RSU r_i. $V[j].p$ represents the computing requirement of b_j. $R_t[i].Q_j^t$ represents the rent-out cost offered by r_i to b_j. Each part of Equation 4.19 is explained as follows:

- $R_t[i].s - (F_j + U_j^t)$ signifies the availability of storage resources. This is calculated by taking the difference of remaining unprocessed data in VM b_j from the remaining storage of RSU r_i. In order to select an appropriate RSU, the predicted data that an application can generate until the deadline is added to the unprocessed data. The generated data can be calculated as:

$$F_j = \sum_{k=t}^{T} \lambda_k \quad (4.20)$$

- Each VM looks for an RSU with a higher DPR because the DPR of RSUs varies. D_i^r represents the DPR of RSU r_i. Lower DPR means higher is the time for completion. The RSU with a higher DPR value is given as the priority.
- The ratio of the RSU's remaining computing resources to the VM's necessary computation requirements is represented by the expression $R_t[i].p/V[j].p$. We employ the best-fit policy to effectively distribute the remaining resources because a VM's need for computing resources is constant.
- Expression $-R_t[i].Q_i^t$ indicates the possible maximum rent-out cost of RSU r_i. Price of the chosen RSU should be minimal in order to reduce the rent-out cost.

4.3.3 Efficient Resource Renting (ERR)

Here, we explain our proposed algorithm which we call ERR given in Algorithm 5. It has two modules - Request_Approval (RA) module and Resource_Aware_Assignment (RAA) module.

- **Request_Approval (RA):** A VM's point of entry into the system is this module. This module is activated when a vehicle carrying a certain amount of data enters the transmission range of an RSU. A VM request is initiated by a vehicle to a nearby RSU. The request is accepted in case any RSU in the network has sufficient resources to store the generated data of requested vehicle. Once the request is accepted, a VM is created in the RSU. There is a possibility that the RSU may not get all of the data that the vehicle generates. The remaining data is then sent in successive time steps in that scenario. If an appropriate RSU cannot be discovered, the request is taken into account in the subsequent time steps.
- **Resource_Aware_Assignment (RAA):** An initial selection of VMs for the assignment is made in this module. The Hungarian Matching algorithm, also known as the Kuhn-Munkres algorithm [54], is then used to assign the selected VMs to the best available RSUs. This is due to the fact that the Hungarian matching algorithm performs better when tackling complicated assignment problems [55]. The problem scenario considered in this work becomes a large-scale assignment problem in a major city scenario due to an increase in vehicles and RSUs. Equation 4.19's considerations for storage resource, computation resource, DPR, and rent-out cost are taken into

account while determining a suitable RSU. Once a suitable RSU has been identified, the VM is moved from that RSU to the destination RSU in accordance with the assignment criteria.

Algorithm 5: Efficient Resource Renting (ERR)

```
1 R_t ← List of RSUs at time t
2 for (t = 0; t < T; t = t + 1) do
3     Request_Approval()
4     Resource_Aware_Assignment()
```

Table 4.1 shows the variables used in ERR. The proposed algorithm ERR is presented in Algorithm 5. It first initializes a list of RSUs to R_t, which stores the information of a set of RSUs. It gets updated at each time step based on two different modules. Those two modules - *Request_Approval*() and *Resource_Aware_Assignment*() are called at each time step till T.

Algorithm 6: Request_Approval()

```
1  M_t[] ← List of all existing VMs at time t
2  M_new[] ← List of newly created VMs
3  M_old^data[] ← List of Old VMs with some data
4  Append temp_dropped to M_new[]
5  for (each VM b_j in M_t) do
6      Update_Storage()
7  for (each VM b_j in M_New ∪ M_old^data) do
8      if (∃j : R_t[i].remaining_s ≥ storage need of VM j and Count_r ! = 0) then
9          Add VM b_j to M_t
10         Count_r = Count_r - 1
11     else
12         Add i to temp_dropped
```

Algorithm 6 admits requests of VMs to the system based on the availability of resources. The algorithm, first initializes the existing VMs, new VMs, and Old VMs. Then add the temporarily dropped VMs to the new VMs lists. It updates the amount of data in all the VMs using M_t. According to the VM's storage requirements, this module once more determines whether a new or an existing VM with data can be assigned to any RSU in the

Table 4.1: *Notations used*

Notations	***Used to***
R_t	At time t, the state of each RSU
M_t	Current state of each VM at time t
M_{new}	List of VMs that are created at t in time
M_{old}^{data}	List of existing VMs with ongoing production of new data till $t-1$
$temp\_dropped$	Temporarily dropped VMs till time step $t-1$
$R_t[i].remaining_s$	Remaining storage in the RSU r_i
$Count_r$	The most channels that can be used to transport data between a vehicle and an RSU
D_i^r	The maximum number of data units that RSU can process per unit time
$M_{t-1}[j].scheduled$	Scheduling status of VM b_j in last time step
$M_t[j].data$	New data status of VM b_j at the beginning of time step t
$M_t[j].s$	The VM's need for storage commencing with b_j at time step t
$M_t[j].data\_amount$	Amount of new data produced for VM b_j at the beginning of time t
$VMAssigned$	List of VMs assigned from the list M_t
ch	VMs from the subset of M_t selected for assignment
$Assigned$	Set of VMs assigned in each iteration using Hungarian algorithm
$R_t[i].v$	Set of VMs in r_i at time step t
vm_edf	List of sorted VMs present in RSU r_i
$vm\_edf[j].completed$	Denotes 1, if VM $vm\_edf[j]$ has completed its execution at the end of time $t-1$, else 0
$vm\_edf[l].c$	Amount of computing need by the VM $vm\_edf[l]$
$R_t[i].remaining_c$	Availability of computing resources in RSU at t
$M_t[j].data\_remaining$	The quantity of data that needs to be processed at time t's beginning
M_{over}	List of VMs completed their execution

system. The VM is introduced to the system if a compatible RSU is discovered; otherwise, it is momentarily dropped. A list called *temp_dropped* contains the dropped VMs.

Algorithm 7: Update_Storage()

```
1 M_t[] ← List of all existing VMs at time t
2 if (M_{t-1}[j].scheduled) then
3     if (M_t[j].data) then
4         M_t[j].s −= D_i^r
5         M_t[j].s += M_t[j].data_amount
6     M_t[j].s −= D_i^r
7 else
8     if (M_t[j].data) then
9         M_t[j].s += M_t[j].data_amount
```

The VM's storage requirements are updated using Algorithm 7 based on knowledge of the most recent scheduling and data generation step. The processed data is deducted from the current VMs' demand if a VM is scheduled in the most recent time step. Similar to this, any new data generated for the VM increases its storage requirements.

Algorithm 8 is the core part of ERR, where, initially the VMs are migrated based on the assignment policy and then executed in the RSU where they are assigned. For the assignment, a set of VMs is chosen from the list M_t, a weight is assigned by using Equation 4.19, and Hungarian matching algorithm is applied to get a suitable match. The process of assignment continues until all the chosen VMs are assigned or no assignment is possible. After the assignment, VMs present in each RSU are sorted based on the earliest deadline first manner. Then, based on the availability of computing resources in the RSU, either the VM is executed or considered in the next time step.

4.3.4 Complexity Analysis

ERR algorithm's time complexity can be calculated using the time complexities of its current modules, *Request_Approval*, and *Resource_Aware_Assignment*. The *Request_Approval* module completes in O(X) time, where X is the total number of vehicles. The module *Resource_Aware_Assignment* takes $O(XY)$ time to complete, where Y is the total number of RSUs. As a result, the temporal complexity of the algorithm ERR is $O(X + XY)$ per time unit.

Algorithm 8: Resource_Aware_Assignment()

```
1  M_t[] ← Incomplete VMs
2  VMAssigned = φ
3  flag = TRUE
4  while (M_t! = VMAssigned and flag == TRUE) do
5      ch = subset of VMs chosen from M_t for assignment
6      W_ji ← ω1(R_t[i].s − (F_j + U_j^t)) + ω2(D_i^r) + ω3(−R_t[i].p/a[j].p) + ω4(−R_t[i].Q_i^t)
7      Assigned = Hungarian_Algo(W_ji)
8      if (Assigned[] == φ) then
9          flag = FALSE
10         continue
11 VMAssigned = VMAssigned ∪ Assigned
12 VMs migrated to the assigned RSUs
13 Update R_t, M_t of RSUs used in assignment
14 for each RSU r_i in R_t do
15     vm_edf ← sort R_t[i].v by earliest deadline first order
16     for each VM b_j in vm_edf do
17         if ((!vm_edf[j].completed) and (R_t[i].remaining_c > vm_edf[l].c)) then
18             M_t[j].data_remaining -= D_i^r
19         else if (vm.edf[j].completed) then
20             Append vm.edf[j] to M_over
21             vm.edf[j].r.remaining_c += vm.edf[j].c
```

The flowchart for the proposed algorithm ERR is given in Figure 4.2.

4.4 Simulation Results

In this Section, we discuss the results obtained from implementing ERR using a Java-based discrete event simulator. A detailed analysis is carried out to measure the end-to-end delay with data partitioning at the end of the section along with the results. A lower Manhattan city scenario [60] with 10 square kilometers of area and a bidirectional road in the city is taken into consideration. There are 9 RSUs placed uniformly in the scenario with a transmission range of 500 meters implying that RSUs are non-overlapping and many areas are not covered by any RSU. Simulation of Urban MObility (SUMO) [18] is used to generate the vehicle's movement. The simulation parameters are taken from [22, 27] and are specified in Table 4.2. We have evaluated ERR for a periodic application, event-driven

Table 4.2: *Simulation parameters for ERR*

Parameters	Values
Vehicles count	50-300
Total simulation time	890
RSU count	9
RSU's per unit base cost	0.001$ - 0.004$
RSU's per unit storage cost	0.001$ - 0.007$
RSU's data processing rate	8-15 Mbps
RSU's per unit computing cost	0.001$ - 0.007$
Per unit data transfer cost	0.002$
VM's fixed migration cost	0.01$
Initial storage needs of VM	100 MB - 300 MB
RSU's transmission range	500 meters
Initial computing need of VM	5 MHz - 40 MHz
RSU's total computing capacity	1000 MHz
RSU's total storage capacity	6000 MB

application, and their combination. A periodic application is one that produces data on a regular basis with a fixed period, whereas an event-driven application only produces data in response to particular events. We have compared the results of ERR with Application Aware Load Balancing (AALB) algorithm [61], Joint algorithm for Selection decision, Computation resource, and Offloading ratio (JSCO) [44], and one Lower Bound (LB) scenario. For the LB scenario, we have considered a fictitious RSU with limitless storage and computing capabilities and a rent-out cost that is the lowest of all RSUs. As a result, all vehicles in the scenario can send any request or data they generate right away to this RSU for processing. We also evaluate ERR with varying weights for values between 0 to 1. To demonstrate a generic case, performance of ERR is shown by giving zero weight to one of the factors and giving non-zero weights to other factors in Equation 4.19. Zero weight signifies that the corresponding factor is ignored while making an assignment decision. Results for periodic, event-driven, and combined applications have been demonstrated in Figures 4.4, 4.6, and 4.8, respectively. Major parameters for simulation are taken from the works [22, 27].

4.4.1 Performance of Periodic Applications

Here, performance of algorithms is shown for periodic applications for different parameters along with the scenario where zero weight is given to one of the factors given in Equation

4.19. Performance of ERR is shown in Figure 4.3 for periodic applications scenario. Figure 4.3a illustrates how VM completion percentage changes with a change in the number of vehicles. It can be observed that percentage of VM completion is higher in the case of ERR as compared to JSCO and AALB. This is because, ERR allocates the VMs to RSUs based on the expected storage needs of VMs (Equation 4.20), thereby resulting in low migrations and a higher completion rate. Figure 4.3b shows the variation of average end-to-end delay with an increase in vehicle count. It can be seen that average end-to-end delay is lesser in the case of ERR as compared to JSCO and AALB. This is due to the fact that ERR uses the best-fit technique for scheduling VMs and the DPR of RSUs for assigning them. Best-fit technique search for RSU with closest available resources. The use of RSU with greater DPR and the best-fit method both help reduce application execution time. Figure 4.3c shows a variation in average rent-out cost with an increase in vehicle count. Average rent-out cost of ERR is lower as compared to JSCO and AALB. This is because, ERR allocates the VMs by considering computing and storage capability, price of the resources, and DPR of RSUs. AALB only focuses on storage and computing resources and JSCO only focuses on computation resources. Figure 4.3d shows a variation of total cost with an increase in the number of vehicles. It can be shown that ERR's overall costs are lower than those of AALB and JSCO. This is because, ERR incurs a smaller number of migrations because of consideration of pricing, along with storage, computing resources, and DPR. The fact that the VM completion percentage, end-to-end delay, rent-out cost, and overall cost are all extremely close to those of the LB scenario shows that ERR is the least expensive option. Overall, it can be seen that the use of rent-out cost and DPR along with remaining storage and computation helps in maximizing the VM completion percentage and minimizing rent-out cost along with the total cost.

Periodic applications - Zero weights

The pricing model given in Equation 4.19 has four weights $\omega_1, \omega_2, \omega_3$, and ω_4. Weights $\omega_1, \omega_2, \omega_3$, and ω_4 assign priorities to storage, DPR, computing resources, and rent-out cost, respectively. To check the performance of our proposed pricing model, we carried out experiments by assigning zero weights to each of the parameters used in Equation 4.19.

Figure 4.4b shows average rent-out cost of completed VMs with an increase in the number of vehicles with various weights considered as zero in ERR along with ERR when all weights

are considered. It can be observed that the performance of ERR is better in all the cases as compared to AALB and JSCO algorithms. This is because, ERR considers factors such as available storage and computing resources, DPR, and rent-out cost for VM allocation. The rent-out cost for ERR when $w_4 = 0$ is more as compared to the other three cases as the chances of VM assignment to those RSUs with more rent-out cost increases in this case. The average rent-out cost with $w_1 = 0$ is higher as compared to the case where computation and data processing of RSU are ignored. This is because, when storage factor is ignored during the assignment, a VM may get assigned to different RSUs having lesser available storage resources leading to more migrations and higher rent-out costs. Figure 4.4c shows average total cost of the completed VMs with an increase in the number of vehicles. Total cost is the summation of rent-out cost and migration cost. The trends show that ERR performs better than JSCO and AALB even when different weights are considered as zero. The migration cost with $w_4 = 0$ is lesser as compared to the other cases where one of the weights is ignored at a time. This is because, when rent-out cost is ignored, the algorithm selects the best RSU by considering only storage resources, computation resources, and DPR required for execution of the VM resulting in a lower number of VM migrations. The migration cost is more in the case of $w_1 = 0$. This is because, by ignoring storage factor, a VM may get assigned to RSUs with lesser available storage leading to more migrations. Due to the same reason the trend in ERR in Figure 4.4a can be observed.

4.4.2 Performance of Event-driven Applications

In this Subsection, performance of algorithms is shown for event-driven applications with different parameters as shown in Figure 4.5. It can be seen in all the figures (Fig.4.5a - 4.5d) that ERR performs better than all the other algorithms as well as the LB scenario. This is because, ERR considers factors such as available storage, computation, DPR, and rent-out cost for assignment.

It can also be observed that performance of ERR for periodic application scenarios is better than event-driven application scenarios. This is because, a huge amount of data is generated during an event, and number of VM migrations increases to accommodate extra data generated. This leads to an increase in end-to-end delay, a decrease in VM completion percentage, and an increase in total cost. Even though a huge amount of data is generated

from different events ERR is able to perform better, this shows how efficiently ERR performs. It is able to complete execution of more number of VMs, less rent-out cost, and total costs even in peak load scenario.

Event-Driven applications - Zero Weights

In Figure 4.6, ERR is evaluated with assigning zero weight to one of the weight factors at a time. Figure 4.6a displays the effect of change in number of vehicles on percentage of VM completion for each weight taken as zero at a time along with ERR with all the weights $(\omega_1, \omega_2, \omega_3, \omega_4)$ taken together. It can be observed that the trends are similar to that of periodic applications. However, the percentage of VM completion is lesser as compared to the periodic application scenario. This is because, a huge amount of data is generated with the occurrence of events leading to unavailability of RSU resources. This increases the overall delay. Due to the same reason a similar performance is there in Figures 4.6b and 4.6c.

4.4.3 Performance of Periodic and Event-driven Applications

In this Subsection, performance of algorithms is shown for both periodic and event-driven applications with different parameters as shown in Figure 4.7. It can be seen in all the figures (Fig. 4.7a - 4.7d) that ERR performs better than all the other algorithms as well as the LB scenario even though more data is generated in this case. This is because, ERR considers factors such as available storage, computation, DPR, and rent-out cost for assignment. Overall, it can be observed that the use of rent-out cost and DPR along with remaining storage and computation helps in maximizing the VM completion percentage and minimizing the rent-out cost and total cost even in case of periodic and event-driven scenarios.

Periodic and Event-Driven Applications: Zero Weights

Here, we show the outcome of ERR for a combination of periodic and event-driven applications when one of the weight factors is assumed to be zero. Figure 4.8a shows a variation of VM completion percentage with the number of vehicles for each weight taken as zero (as mentioned in Section 4.4) at a time along with ERR with all the weights taken together. According to the trends, the percentage of VMs that complete their tasks drops dramati-

cally when compared to event-driven and periodic applications separately. This is because, a huge amount of data is generated when periodic and event-driven applications are combined together. Figure 4.8b shows rent-out cost for a combination of periodic and event-driven applications with an increase in the number of vehicles. The rent-out cost shows a significant increase in data generation with an increase in number of vehicles. This is because of more data generation for this type of application which implies that VM waits in the RSUs for a longer duration resulting in increased rent-out cost. ERR outperforms JSCO and AALB. Figure 4.8c shows the variation of average total cost with respect to the number of vehicles for a combination of periodic and event-driven applications. The trends show that the total cost and rent-out cost of ERR is similar. This is because, ignoring rent-out costs at the time of assignment of VMs to the RSUs may lead to a lesser number of migrations.

4.5 Delay Analysis

Delay is an important parameter in VANETs. Here, we are focusing on end-to-end delay. It is defined as the duration of data processing of a request generated by a vehicle. It comprises four major components - 1) the time taken to transfer data from the vehicle to RSU, 2) the duration for which a VM migrates from one RSU to another, 3) the duration for which a VM waits in any RSU for getting the computing resources or the duration for which a VM waits for arrival of data from the vehicles, and 4) the duration of VM execution in an RSU. The waiting time may vary depending on application types. Waiting time is more in case the period is larger in the case of periodic applications or inter-event duration is larger in the case of event-driven applications. The different components of end-to-end delay are mentioned below.

1. *Transfer delay:* It is defined as the duration for transferring data from a vehicle to a neighboring RSU.

2. *Migration delay:* It is the sum of the time taken to migrate a VM along with its associated data from one RSU to another RSU till the deadline of the corresponding applications.

3. *Waiting delay:* It is the duration for which- (a) a VM waits in an RSU to receive data from vehicles, and (b) a VM waits for computation resources from the RSU to execute the received data from vehicles.

4. *Execution delay:* It is the duration for processing the data present in a VM at an RSU.

Measurement of end-to-end delay for each portion of data can help to analyze performance of the algorithm for randomly incoming data. This is why it is necessary to partition the data into small parts. Partitioning data into small parts helps in scheduling the VM requests in the RSUs. We consider four states for the generated data by a vehicle. These are – 1) Transfer state, 2) Migration state, 3) Waiting state, and 4) Execution state.

4.5.1 Data Partitioning

Identification of data that remains in one of the aforementioned states is important to analyze the end-to-end delay. We partition the data into fixed-sized blocks that we call chunks to analyze the performance of ERR for randomly incoming data. Each chunk of data is identified by one identification number which helps in maintaining the order of chunks.

Let us assume that there are m chunks of data generated at vehicle v_j and processed by a VM b_j using the resources of some of the RSUs present in the scenario. Chunks are represented by $C_1^j \ldots C_m^j$. $Data_j$ is the data generated by the vehicle v_j and is calculated as follows:

$$Data_j = \sum_{k=1}^{m} C_k^j \tag{4.21}$$

4.5.2 Calculation of Delay

We calculate end-to-end delay of each request as per Equation 4.22. End-to-end delay D_j is calculated as the summation of the delay incurred for each chunk of data at different RSUs and the VM's waiting time in an RSU. D_j is defined below:

$$D_j = \sum_{i=1}^{Y} \left(\sum_{k=1}^{m_i} d_{C_k^j}^i + \zeta_j^{i,n} \right) \tag{4.22}$$

$$d_{C_k^j}^i = d_k^t + d_k^m + d_k^w + d_k^e \tag{4.23}$$

where d_k^t is the data transfer delay for chunk C_k, d_k^m is the migration delay for chunk C_k, d_k^w is the waiting delay for chunk C_k, d_k^e is the execution delay for chunk C_k, and $\zeta_j^{i,n}$ is

the waiting delay of VM b_j at RSU r_i for event number n. $\zeta_j^{i,n}$ is modeled as gamma distribution [62] with probability density function represented by Equation 4.24. Gamma distribution predicts the waiting time of the n^{th} event. Therefore, Gamma distribution is chosen to analyze the waiting delay of the VM. The waiting time of the VM varies between 0 and T. In our scenario, we assume that the occurrence of an event follows the memoryless property in the case of event-driven applications. In the case of periodic applications, data gets generated based on the period.

$$f_{\zeta_j^{i,n}}(t) = \frac{\pi^n}{n!} t^{n-1} e^{-\pi t}, 0 \leq t \leq T \tag{4.24}$$

Here, π is the arrival rate, n is the event number, and f is the probability density function as given in Equation 4.24.

4.5.3 Analytical Results

Data partitioning helps to analyze the end-to-end delay for each chunk of data. In case a chunk of data is available with the RSU, it gets processed and does not wait for the next chunk of data. The end-to-end delay is calculated for each chunk of data along with the VM's waiting time. We have analyzed the end-to-end delay with data partitioning. The results obtained from simulation without data partitioning and the analytical data obtained with data partitioning are shown in Figure 4.9. Data partitioning helps to measure the end-to-end delay of ERR effectively for different types of applications with variations in incoming data. Figures 4.9a, 4.9b, and 4.9c show the performance of periodic applications, event-driven applications, and their combination, respectively both with and without data partitioning. The analytical results with data partitioning and the results obtained from the simulations without data partitioning follow similar trends. However, the delay in case of data partition is lesser as compared to the no data partition scenario as shown in Figure 4.9. This is because, the chunks of data are scheduled even with the availability of a small amount of resources.

4.6 Modified Efficient Resource Renting (MERR)

In the previous section, we have partitioned data into chunks, that modify our algorithm ERR and we call that algorithm MEER. Here, each chunk of data can be assigned to

any RSU of the network. This leads to a significant decrease in VM migrations resulting in lower delay and higher VM completion percentage. The core part of ERR i.e. Resource_Aware_Assignment() has been modified to change the assignment procedure as shown in Algorithm 9. Here, in the first iteration, all the data generated by the VM request in one RSU is partitioned and considered as remaining chunks. After scheduling the request the remaining chunks are assigned to some other RSUs. Along with that lock time for the VM in the last assignment is taken into consideration. Lock time is the time for which the VM is locked for rescheduled and never rescheduled. To minimize the number of VM migrations the lock is applied to the assignment. During the lock period, the added chunks to the VM are assigned to other RSUs of the network which minimizes the end-to-end delay and increases the number of VM completion. The overall cost also gets minimized.

4.6.1 Modified Algorithm

The proposed modification applied to ERR and the modified version of the algorithm is specified in Algorithm 9.

4.6.2 Results of MERR

The performance of MERR is shown in Figures 4.10, 4.11, and 4.12 for periodic applications, event-driven applications, and their combinations respectively. In all scenarios, MERR outperforms other algorithms for VM completion percentage, end-to-end delay, rent-out cost, and total cost. This is because, the chunk of data of each VM can be processed in other RSUs in case of availability of resources. This reduces end-to-end delay, and overall cost, and increases VM completion percentage.

Algorithm 9: Modified Resource_Aware_Assignment()

$\varphi_j \leftarrow$ *remaining chunks of VM j*
$t_j^l \leftarrow$ *lock time for* VM_j *for last assignment*
$M_t[] \leftarrow$ *Incomplete VMs*
$VMAssigned = \phi$
$flag = TRUE$
while $(M_t! = VMAssigned$ *and* $flag == TRUE)$ **do**
    $ch =$ subset of VMs chosen from M_t for assignment
    $W_{ji} \leftarrow \omega_1(R_t[i].s - (F_j + U_j^t)) + \omega_2(D_i^r) + \omega_3(-R_t[i].p/a[j].p) + \omega_4(-R_t[i].Q_i^t)$
    $Assigned =$ Hungarian_Algo(W_{ji})
    Apply lock on assigned VMs
    if $(Assigned[] == \phi)$ **then**
        $flag = FALSE$
        continue
$VMAssigned = VMAssigned \cup Assigned$
VMs migrated to the assigned RSUs
Update R_t, M_t of RSUs used in assignment
for *each RSU* r_i *in* R_t **do**
    $vm\_edf \leftarrow$ *sort* $R_t[i].v$ by earliest deadline first order
    for *each VM* b_j *in vm_edf* **do**
        if $((!vm\_edf[j].completed)$ *and* $(R_t[i].remaining_c > vm\_edf[l].c))$ **then**
            $M_t[j].data\_remaining$ -= D_i^r
        else if *(vm.edf[j].completed)* **then**
            Append $vm.edf[j]$ to M_{over}
            $vm.edf[j].r.remaining_c$ += $vm.edf[j].c$
        Assign the remaining chunks of each VM to another RSU

4.7 Chapter Summary

This chapter proposed a pricing model that addresses the problem of resource renting within the RSU network. A resource renting algorithm is proposed which takes care of rent-out cost, available storage, available computation, and data processing rate. To further optimize the approach, the tasks are partitioned into chunks before doing the request assignment. ERR increases the VM completion percentage by approximately 61.82%, decreases the delay by 62.5%, and decreases the total cost by 80% compared to JSCO for 150 vehicles for periodic applications.

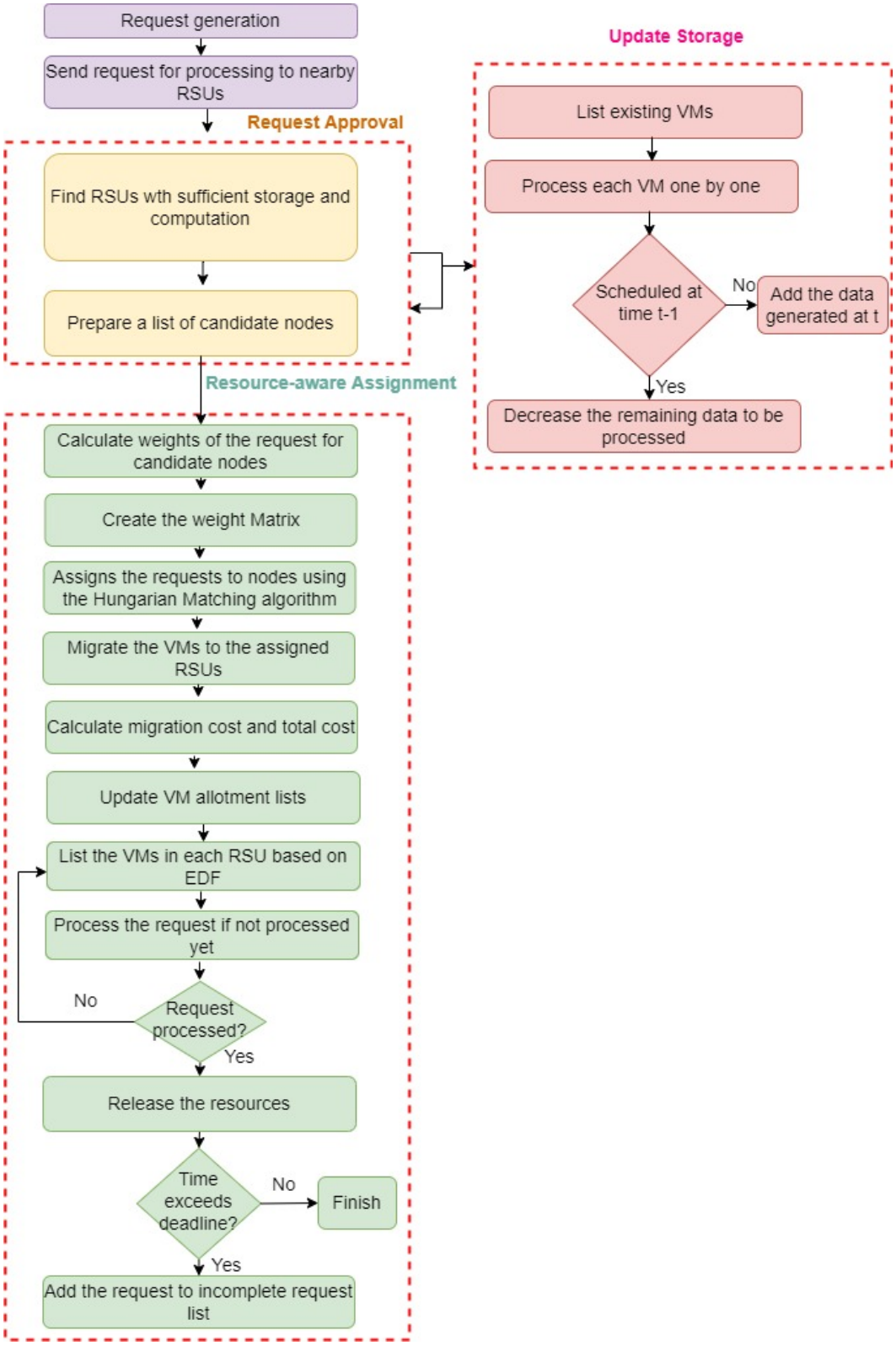


Figure 4.2: *Flow chart for ERR*

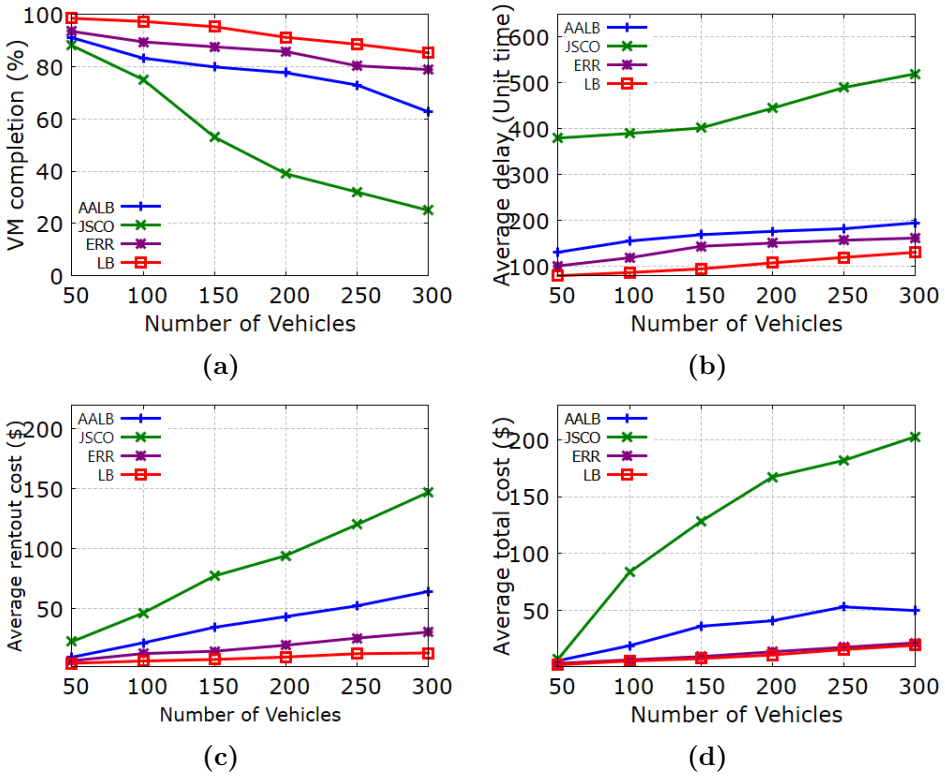


Figure 4.3: *Performance of algorithms for periodic applications*

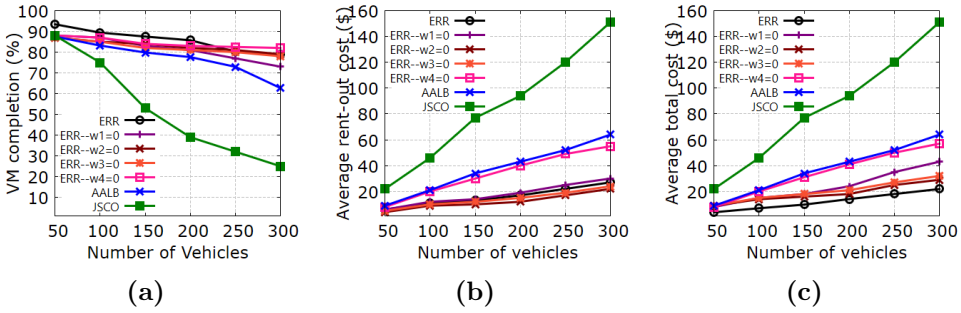


Figure 4.4: *Performance analysis after assigning zero weights to different parameters for periodic applications*

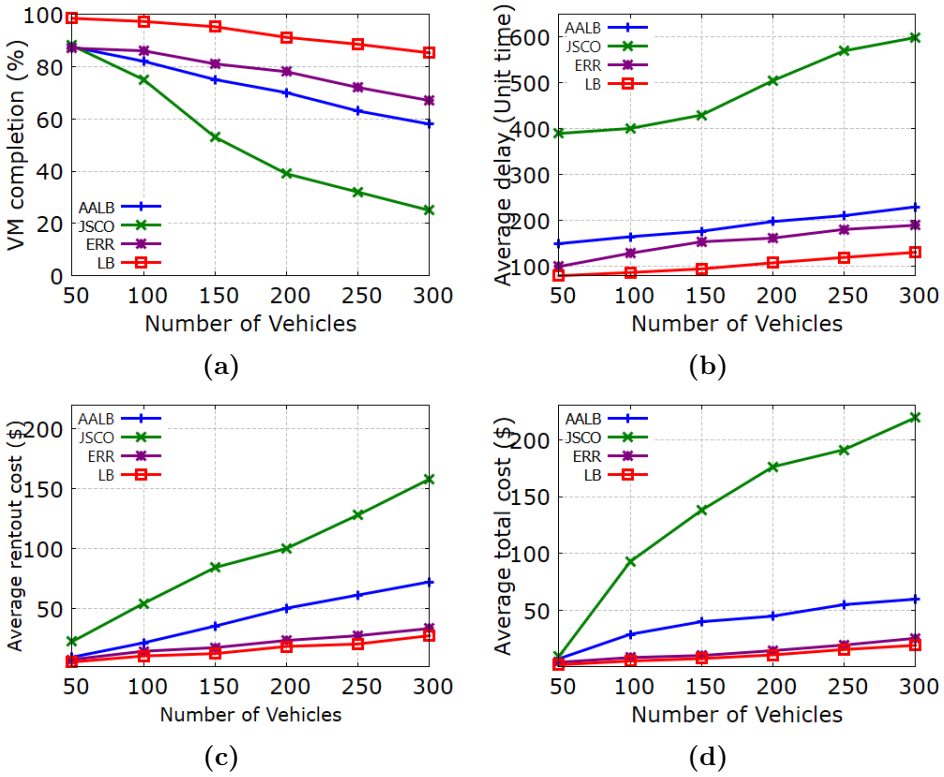


Figure 4.5: *Performance of algorithms for event-driven applications*

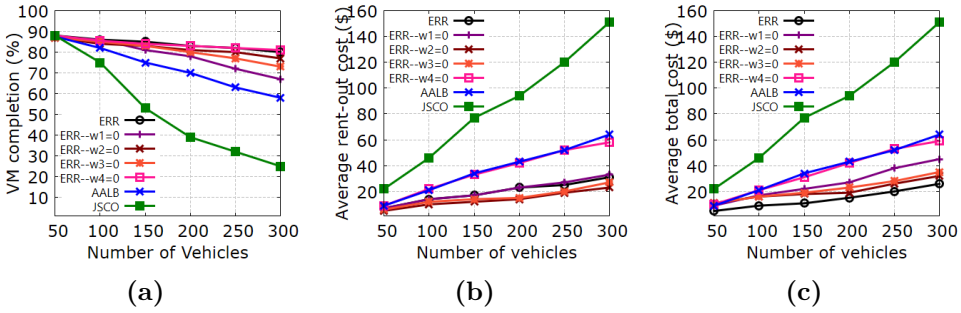


Figure 4.6: *Performance analysis after assigning zero weights to parameters for event-driven applications*

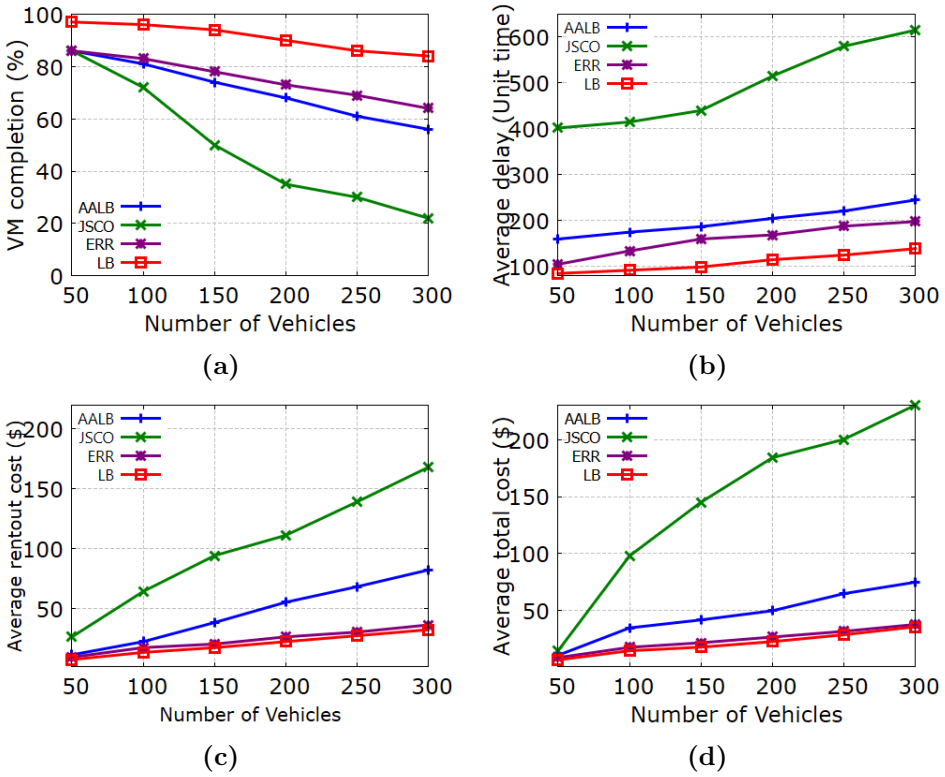


Figure 4.7: *Performance of algorithms for a combination of periodic and event-driven applications*

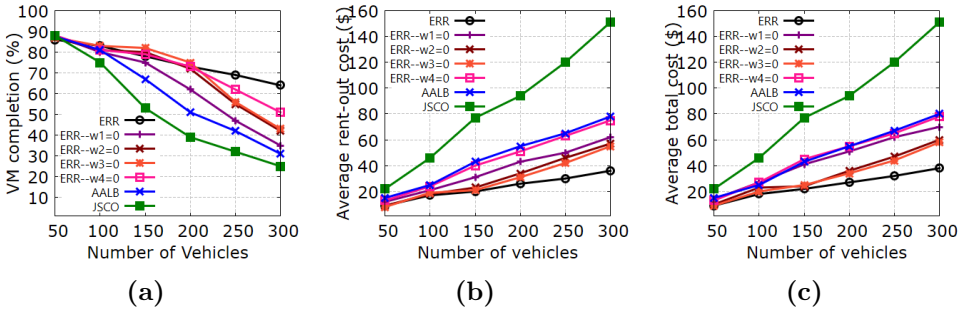


Figure 4.8: *Performance analysis after assigning zero weights to parameters for a combination of periodic and event-driven applications*

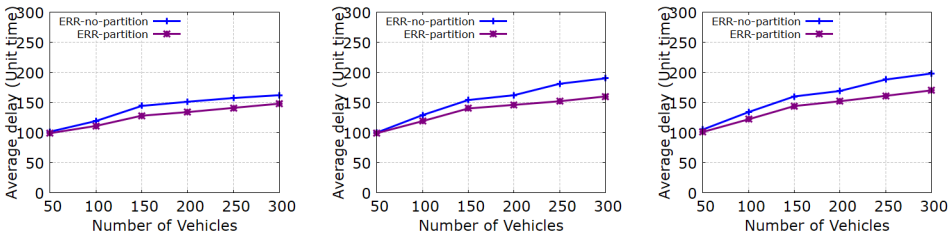


Figure 4.9: *Trends of delay for (a) periodic applications, (b) event-driven applications, and (c) their combination*

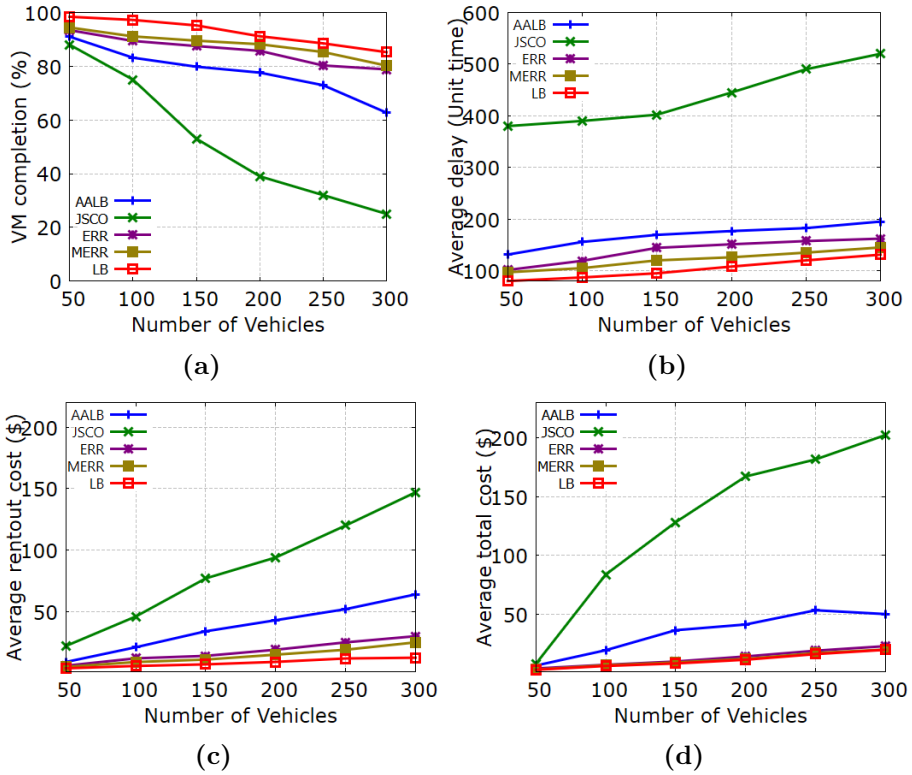


Figure 4.10: *Performance of algorithms for periodic applications*

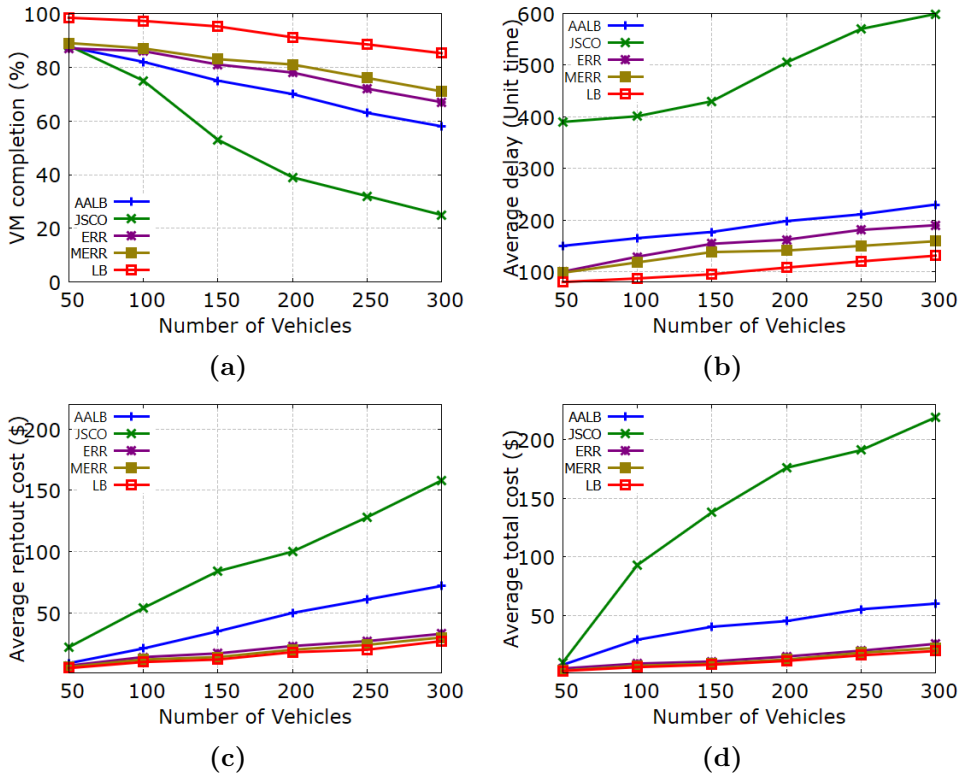


Figure 4.11: *Performance of algorithms for event-driven applications*

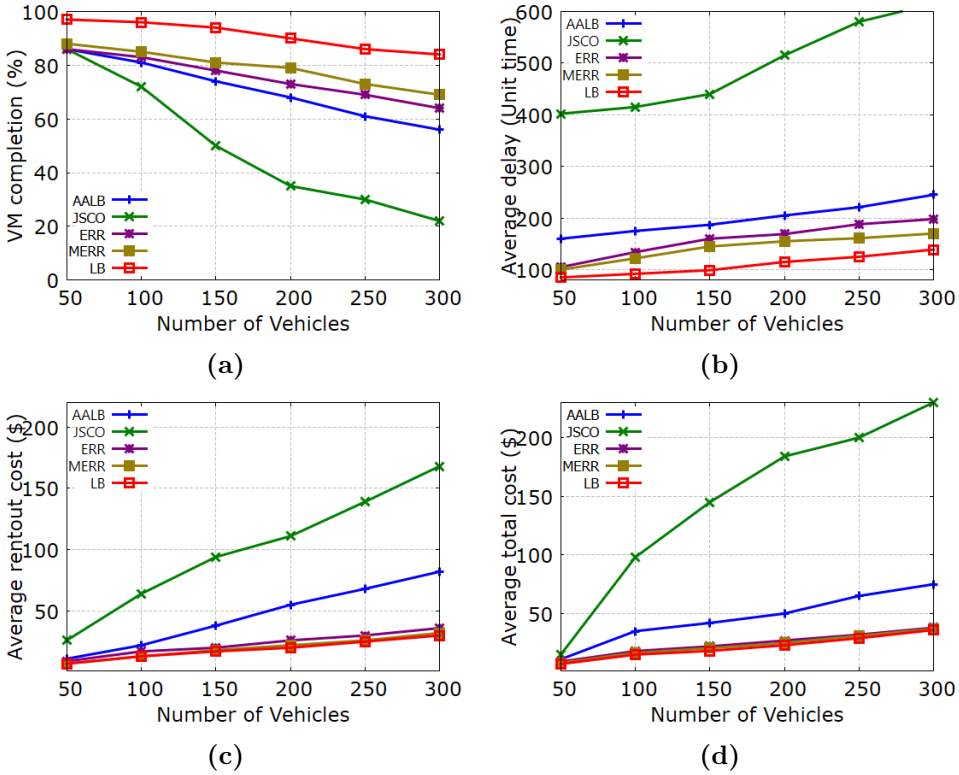


Figure 4.12: *Performance of algorithms for a combination of periodic and event-driven applications*

5

Profit Maximization in Heterogeneous Vehicular Networks

In this chapter, we have proposed an algorithm for maximization of profit for SPs in heterogeneous vehicular networks while maintaining the QoS for processing the application requests. To increase resource availability, PVs and CC are considered as options for processing the service requests.

5.1 Introduction

Vehicles in the city scenarios are unevenly distributed. This distribution is due to the location of the city and the time of the day. Vehicles use multiple applications, each with different resource requirements and deadlines. Due to the huge number of requests, RSUs sometimes get overloaded. This leads to high chances of service failure and a decrease in QoS. Deployment of more RSUs to the city is one of the possible options for avoiding this overloaded scenario. However, due to the huge deployment and maintenance costs of RSUs, it is not economical to cover the entire city with RSUs. PVs are considered as an economically viable option for processing the requests. However, due to its limited resource capacities, PVs can not be used for all request types. In this scenario, CC is considered as

one option for processing the requests. This motivates us, to propose better solutions that can handle overloaded scenarios.

5.2 System Model

In this Section, we describe a system model which includes all the network components. We

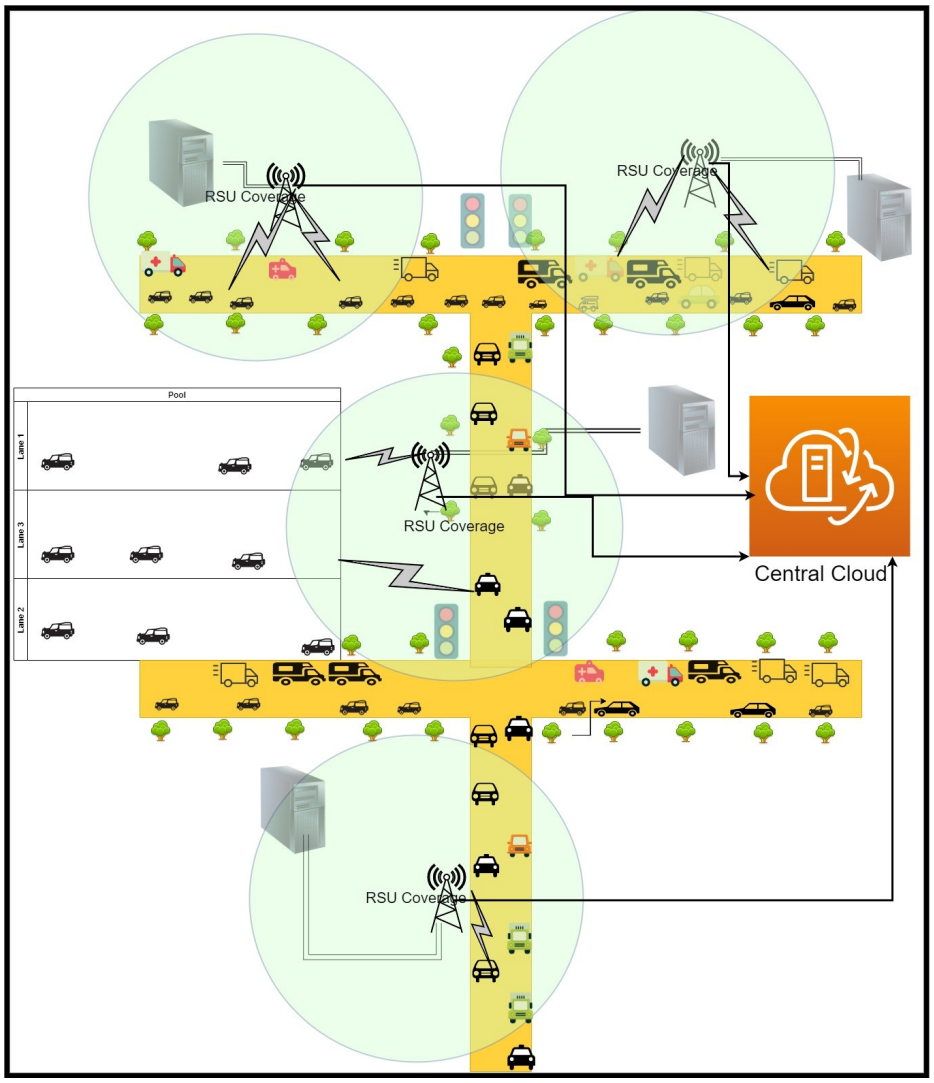


Figure 5.1: *System model*

consider a city scenario with an area of 10 square kilometres where nine RSUs are placed uniformly. We consider three SPs, each of which consists of three RSUs. SPs accept requests

from the on-road vehicles through one RSU. Each vehicle runs several applications. Each application generates single resource requests based on the user's preference. The average data generation rate for each application is known. Data generated by an application is temporarily stored in the vehicles, which need to be processed within the deadline of application. The SPs coordinate with other SPs along with the CC and PVs to process the requests. There is a fixed number of parking lots in the city, where vehicles are parked. All vehicles in one parking lot are within the transmission range of one RSU as shown in Figure 5.1.

5.3 Network Terminologies

In this Section, we defined some terminologies considered in this work.

5.3.1 Requester Vehicles

Requester vehicles are on-road vehicles that have some resource requirements. These vehicles have a fixed route and run one or more applications. Each application generates one resource request. The request includes storage resources, computing resources, and content delivery time.

5.3.2 Road Side Units (RSUs)

RSUs are static nodes placed in the city along the side of roads. These are equipped with computing and storage capabilities to provide services to the requester vehicles. These RSUs serve as intermediate nodes between vehicles and the CC, helping to reduce latency and network congestion by assigning the requests to either CC or PVs. RSUs can perform tasks such as data aggregation, processing, and analysis of data generated by vehicles.

5.3.3 Parked Vehicles (PVs)

PVs are stationary vehicles located within a specific region of a city and fall within the transmission range of an RSU. The information about PVs, such as parking duration, available storage and computation is communicated to the nearby RSUs continuously. This helps SPs to make decisions about request assignments. Assigned requests are processed within the parking duration. This arrangement allows PVs to actively participate in the processing

and transmission of data, leveraging their available resources and contributing to the overall service provisioning ecosystem.

5.3.4 Central Cloud (CC)

A CC is a distant data centre utilized for storing, processing, and managing data and applications. It serves as a platform that offers various services, including real-time traffic information, route planning, and other data-intensive operations. The robust infrastructure and extensive storage capacity enable CC to execute large-scale data processing tasks efficiently. It acts as a centralized hub, facilitating the delivery of critical services that require substantial computational power and storage capabilities. CC plays a major role in providing scalable and high-performance solutions for diverse applications and services in the networked environment.

5.3.5 Service Federation

Service federation is a process by which service requests are processed by renting resources from other entities. Here, the SPs establish interconnections with one another, as well as with PVs and the CC. Through this network of RSUs, SPs facilitate communication and data exchange among themselves, PVs, and the CC. This interconnected infrastructure allows for seamless connectivity and efficient management of resources and services provided by the SP to various entities. The process involves SPs providing services to requester vehicles by utilizing resources from other SPs, PVs, and the CC with guaranteed QoS. This collaboration allows SPs to leverage the combined resources and capabilities of multiple entities, ensuring efficient service delivery and maintaining the desired level of performance. In case a request is made to an SP, it is necessary to assign the request to an appropriate node out of CC, PVs other SPs. This increases the chance of maximizing profit for the SPs.

5.4 Request Assignment: An Economic Perspective

Vehicles generate requests that are transferred to nearby RSUs for processing and assignment based on optimal node assignment policy. To ensure completion of request processing, RSUs, PVs, or CC are assigned the request by the SPs. Cost Model (CM) and Selection Criteria (SC) are considered before the assignment of requests.

5.4.1 Cost Model (CM)

The cost associated with each resource type is specified in the CM. To process a request a vehicle may pay the cost to RSUs or CC or PVs. There are four types of costs – 1) cost of processing, 2) cost of storage, 3) cost of computation, and 4) cost of content delivery.

5.4.1.1 Cost of Processing

It is a fixed cost for processing a request by an SP. This cost is applicable each time when a request is processed. Payment is received by the RSU which takes the service request from vehicles. Cost of processing is only charged by SPs not by a PV or CC.

5.4.1.2 Cost of Storage

It is a fixed cost charged by all types of nodes (RSUs, PVs, and CC). Each of these three nodes has a different rent-out cost. This makes the scenario complex for the SPs to assign the request to one of these nodes.

5.4.1.3 Cost of Computation

Computation cost of CM focuses on per-unit computing cost. Each type of node has a different cost profile. In case an RSU does not have sufficient resources, it rents resources from other RSUs or PVs, or CC.

5.4.1.4 Cost of Content Delivery

Content delivery services in VANETs can be designed to support different types of information dissemination, such as safety messages, traffic updates, or multimedia content including video streaming. Content delivery service allows the SPs to provide the service to vehicles by utilizing the resources of RSUs of other SPs, PVs, or CC. This is done to provide uninterrupted services without affecting the QoS of SPs.

5.4.2 Selection Criteria (SC)

There are three types of nodes used to assist the SPs - 1) parked vehicles, 2) central cloud, and 3) RSUs of other SPs. It is important to select those nodes in such a way that more number of requests get processed with minimum cost.

5.4.2.1 Parked Vehicle Selection

Selection of PVs should be done based on available resources and "time of stay" in a particular parking lot. Probability of PVs remaining in the parking lot is too high which avoids the situation of the sudden departure of PVs. Each PV stays in the parking lot for a fixed duration. This information is intimated to the nearby RSUs when a vehicle enters to the parking lot. Each PV is an independent entity. It gets paid for the resources it provides to the SPs.

5.4.2.2 Central Cloud Selection

CC is the most suitable option for applications that are delay-tolerant and require a huge amount of resources. The time of uploading a task to a CC takes a significant amount of time that depends on the amount of data and network bandwidth. The applications that require less interaction with servers or applications that generate huge amounts of data are most suitable for CC in overloaded scenarios.

5.4.2.3 RSU Selection

RSUs are the nodes connected to the current node via the backbone network. They have resource capacities larger than the PVs and lesser than the CC. Applications that require requests to be processed with minimum delay with moderate computation required can be assigned to the RSUs. During assignments, rent-out cost of RSUs belonging to the same SP is considered as zero.

5.4.3 Strategy Determination

The request received by an RSU needs assignment to a node by which SP gets maximum profit without affecting the QoS. Thus, it is necessary to have an approach for selecting a suitable node out of the available options. There are three options for processing the requests - 1) at an RSU of other SPs, 2) at the CC, and 3) in a PV. The RSU's DPR is lesser as compared to CC but greater than PVs. The PVs have less resource capacity than other options. However, the cost and transmission delays are lesser in the case of PVs than in other node types. In scenarios where none of the RSUs and PVs are able to serve the request or requests with huge resource requirements, CC is chosen as the option. Thus, determining whether a request is assigned to PVs or other RSUs or CC is challenging. A

decision is made based on application types, resource requirements, deadlines, and available resources for which maximum profit is generated by the SPs without affecting QoS.

5.5 Problem Formulation

Let T be the total time considered for simulation. Let there be N number of SPs represented by the set $\mathcal{S} = \{s_1, s_2, ...s_N\}$. SPs have different numbers of RSUs represented by the variable Γ. Each set of RSUs in an SP is represented by $\mathcal{E} = \{r_1, r_2, \ldots, r_\Gamma\}$. We represent an RSU r_i by the tuple $\langle C_i, S_i, D_i, P_i^s, P_i^c, P_i^d \rangle$, where C_i is the computing capacity, S_i is the storage capacity, D_i is the DPR, P_i^s is the per unit rent-out cost, P_i^c is the per unit computation cost, and P_i^d is the per unit content delivery cost.

Let there be X vehicles represented by the set $\mathcal{V} = \{v_1, v_2, \ldots, v_X\}$. Each vehicle $v_j \in V$ is represented by a tuple $\langle P_j, a_j \rangle$, where

- P_j is the path followed by v_j represented as a sequence of RSUs $\langle r_1^j, r_2^j, \ldots, r_k^j \rangle$, where k is a variable whose value depends on the number of RSUs covered by the vehicles till the end of the journey
- a_j is the total number of applications run by vehicle v_j.

Let there be H number of total requests generated by all vehicles of the system represented by the set $\mathcal{Q} = \{q_1, q_2, \ldots, q_H\}$. Each request q_k is represented by a tuple $\langle s_k, c_k, t_k, \mathcal{L}_k, \wedge_k^s, \wedge_k^c, \wedge_k^t \rangle$, where

- s_k is the storage requirement of request q_k
- c_k is the computing requirement of request q_k
- t_k is the content delivery duration requirement of request q_k
- $\mathcal{L}_k$ is the deadline of request q_k
- $\wedge_k^s$ is the per unit storage cost that a vehicle pays for request q_k to any SPs
- $\wedge_k^c$ is the per unit computing cost that a vehicle pays for request q_k to any SPs
- $\wedge_k^t$ is the cost per unit time of content delivery a vehicle pays to an SP.

Let there be B number of PVs that rent their resources to one of the SPs represented by the set $\mathcal{Z} = \{z_1, z_2, \ldots, z_B\}$. z_l is represented by the tuple $\langle c_l, s_l, d_l, r_l \rangle$, where c_l is the computing capacity, s_l is the storage capacity, d_l is the DPR, and r_l is the index of associated RSU to which z_l can rent it's resource.

The outputs are represented by three sets of variables $\{x_{ijt}\}$, $\{y_{ijt}\}$, and $\{z_{ijt}\}$, $\forall r_i \in \mathcal{E}$, $\forall v_j \in V$, and $\forall t, 1 \leq t \leq T$. Request generated by an application of vehicle v_j is represented by the output variables, which are:

- x_{ijt} is set to 1 if the request is assigned in RSUs, PVs or CC at time t, otherwise 0
- y_{ijt} is set to 1 if the request is scheduled in RSUs, PVs or CC at time t, otherwise 0
- z_{ijt} is set to 1 if the request is executing in RSUs, PVs or CC at time t, otherwise 0.

Let ζ_{jt} be the vehicle's position. It is set to 1 if a vehicle is present in the parking lot otherwise it is 0. Let I be the total number of incomplete requests at a time t. We also define indicator variables F_j^r, F_j^p, and F_j^c to record the finishing node. Value of F_j^r, F_j^p, and F_j^c is 1 indicates that the request finishes its processing by RSUs, PVs, and CC respectively, otherwise, the value is 0. Let I_j be the total number of incomplete requests.

5.5.1 Assumptions

- SPs provide services such as data storage, computation, and content delivery to vehicles.
- Route of a vehicle is fixed i.e. set of RSUs the vehicle passes through in the lifetime of its applications is always the same.
- There is a delay in the transfer of data from the vehicles to RSU and between the RSUs.
- Application running in vehicles consumes all its resources while running on roads.
- A vehicle present in the parking lot uses some memories for running basic applications at the back end.
- Requests generated in vehicles may wait in the vehicle's queue in case of unavailability of RSUs.

- A request can be processed by at most one of the nodes from RSU, CC, or PVs at a time.
- The resource used by a request can not exceed the available resources at RSUs, PVs, and CC at any instant.

5.5.2 Constraints

- The sum of the number of requests completed their processing by RSUs, PVs, and CC along with the number of incomplete requests is less than equal to the total number of requests generated by all vehicles.

$$\sum_{j=1}^{H} \left(F_j^r + F_j^p + F_j^c + I_j \right) \leq Q \tag{5.1}$$

- Total execution time required by a request generated by the applications of vehicle v_j is less than departure time of vehicle from the last RSU in its path.

$$\sum_{t=0}^{T} z_{ijt} \leq \mathcal{L}_j, \forall v_j \in V \tag{5.2}$$

- Each request must be assigned to at most one RSU or one PV or CC at a time.

$$r_i \in \mathcal{E}, t \in T, x_{ijt} <= 1, \forall v_j \in V \tag{5.3}$$

- One SP can not assign a request to PVs of other SPs.

$$x_{ijt} < 1, \forall r_i \in \mathcal{E} \ \ \& \ \ z_i \in \mathcal{Z}, \forall q_i \in Q, z_i \notin \mathcal{E} \tag{5.4}$$

5.5.3 Calculation of Profit

The profit is calculated by subtracting expenditure towards infrastructure cost, rent-out cost, and penalty cost from service cost. Revenue is calculated by multiplying the unit price for resources and the time of use of those resources. Resource utilization plays an important role in the maximization of profit for SPs. Thus, the duration for which SPs use their resources should have to be maximized to maximize profit. The infrastructure cost is fixed

for an RSU. Rent-out cost is dependent on the time for which an SP uses the resources of other RSUs or PVs of other SPs or CC.

Let us assume that E_{til} is the profit generated by the SP s_i, which can be defined as:

$$E_{til} = P_{til} - \left(cost_{til}^{infra} + cost_{til}^{rent} + cost_{til}^{pen}\right) \tag{5.5}$$

$$P_{til} = P_{til}^{p} + P_{til}^{s} + P_{til}^{c} + P_{til}^{d} \tag{5.6}$$

$$cost_{til}^{rent} = \sum_{i=1}^{N}\sum_{j=1}^{\Gamma} cost_{tij}^{rent} + \sum_{l=1}^{B} cost_{til}^{rent} + cost_{tic}^{rent} \tag{5.7}$$

$$cost_{til}^{pen} = \sigma_1 \times \gamma_{til}^{s} + \sigma_2 \times \gamma_{til}^{c} + \sigma_3 \times \gamma_{til}^{d} \tag{5.8}$$

where P_{til} is the cost of using the resources at time t for the vehicle, P_{til}^{p} is the processing cost for using the service from the SP at time t, P_{til}^{s} is the per unit storage cost, P_{til}^{c} is the per unit computing cost, P_{til}^{d} is the per unit content delivery cost, $cost_{til}^{rent}$ is the rent-out cost incurred by RSU i by using the resources of RSU k at time t, $cost_{til}^{rent}$ is the rent-out cost incurred by RSU i by using the resources of PV l from the set of PVs B at time t, $cost_{til}^{rent}$ is the rent-out cost incurred by RSU i by using the resources of CC, and $cost_{til}^{pen}$ is the penalty cost incurred for violating the Service Level Agreement (SLA) at time t.

In Equation 5.8, σ_1, σ_2, and σ_3 are penalties per unit of storage, per unit computing, and per unit content delivery time, in case of failure in SLA. γ_{til}^{s} is the amount of data requested for processing by the request q_i, γ_{til}^{c} is the amount of computation requested by the request q_i, and γ_{til}^{d} is the content delivery duration requested by q_i.

5.5.4 QoS Measurement

The applications running within the vehicles possess distinct QoS parameters such as delay, jitters, and others. The SP must ensure the fulfilment of these QoS requirements. The specific details regarding these QoS parameters are documented in the SLA established between the requesting vehicle and the SP. The SLA serves as a contractual agreement that outlines the agreed-upon levels of service quality. In case an SP fails to meet the obligations outlined in the SLA, penalties are enforced upon SPs. These penalties are designed to be more significant than the actual service cost. The purpose of imposing penalties is to hold the SP accountable for any shortcomings or failures in meeting the agreed-upon QoS standards.

By implementing penalties that exceed the service cost, there is a strong indication for the SP to prioritize and maintain the specified QoS. Penalties imposed on an SP for SLA violations serve multiple purposes. Firstly, discouraging the SPs from neglecting or disregarding the QoS requirements of applications running in the vehicles. Secondly, penalties act as a form of compensation for requesting vehicles, compensating them for the inconvenience or negative impact caused by service failure. It helps to ensure that requesting vehicles receive proper compensation in cases where the QoS expectations are not met.

5.5.5 Problem Definition

The problem we address is to schedule the requests to one of the destination nodes in such a way that profit of SPs is maximized and number of requests processed is maximized with QoS guarantee subject to the constraint given in Section 5.5.2. Hence, the objective functions are stated as

$$maximize \sum_{t=1}^{T} \sum_{i=1}^{N} \sum_{l=1}^{\Gamma} \left(P_{til} - \left(cost_{til}^{infra} + cost_{til}^{rent} + cost_{til}^{pen} \right) \right) \tag{5.9}$$

$$maximize \sum_{t=1}^{T} \sum_{i=1}^{H} \left(F_{ti}^{r} + F_{ti}^{p} + F_{ti}^{c} \right) \tag{5.10}$$

In this work, we consider an optimization problem that corresponds to a matching problem, where a set of requests need to be assigned to a set of nodes (RSUs, PVs, CC). The potentially large number of vehicles and RSUs makes the scenario a large-scale assignment problem. The objective is to make the right decision to maximize the number of requests processed is an optimization problem. The RSUs have limited capacity and the requests have to be processed with QoS guaranteed is similar to the patient hospital problem, which is NP-hard [63]. Thus, our problem becomes an NP-hard problem. We propose a heuristic algorithm to optimally assign the vehicle request to one of the nodes (RSU, PV, or CC) by which a maximum number of requests gets processed within the lifetime and maximizing the profit of the SPs.

5.6 Proposed Methodology

In this Section, we propose an algorithm that we call Adaptive Algorithm for Profit Maximization (AAPM). Profit of the SPs is maximized by the efficient assignment of requests to

a node from RSUs, CC, and PVs. Once the vehicle comes in the transmission range of an RSU and the vehicle has some request to be processed, a request is sent to the corresponding RSU of a SPs. Then the SP coordinates with other SPs, CC, and PVs before deciding where to assign the request. Flow chart of the proposed methodology is given in Figure 5.2. Proposed algorithm AAPM has three different modules:

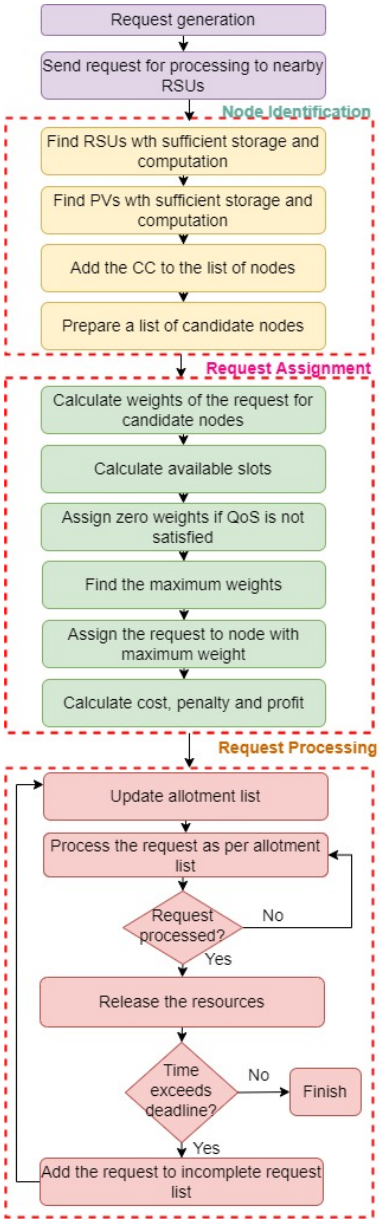


Figure 5.2: *Flow chart of AAPM*

1. Node Identification: This module aims to identify nodes that have sufficient available resources by evaluating the remaining resources. It focuses on selecting nodes that possess equal or greater amounts of resources for the assignment process. By doing so, this module ensures that only capable nodes are considered. This increases the likelihood of successful assignments. This approach helps to optimize request assignment with compatible nodes and ensure efficient utilization of available resources.

2. Request Assignment: The request assignment process involves allocating each request to a specific node for a fixed duration. This duration is crucial because, in a scenario with heterogeneous nodes, each node has its capacity while each request has its resource requirements. Preempting requests frequently during execution can increase context switch time. To minimize context switching, the proposed assignment algorithm adapts a non-preemptive approach. However, a non-preemptive process may increase waiting time for other requests in the queue. Hence, selecting an ideal time slot for a request assignment becomes crucial. The proposed algorithm considers factors such as node types, available resources, and the number of pending requests to determine the optimal slot time for request assignment.

3. Request Processing: Each request from the set of requests is chosen for processing using the round-robin scheduling method. However, the duration for which the requests are processed is as per the time assigned in request assignment phase. In case a request completes its processing before the pre-assigned slot, the resources allocated for that slot are deallocated, which is used by other requests. Requests which take more time to process than their lifetimes are terminated and added to the incomplete request lists.

5.6.1 Calculation of Weight

Weight is a value calculated based on the target node's resource capacity and resource need of a request. To maximize profit of SP, it is necessary to maximize the resource uses. However, in the scenarios where profit is maximized without the use of resources of current SP, is considered in Equation 5.15. Here, the weight equation considers several factors such as preference, type of nodes, SP, remaining available storage, remaining available computations, and rent-out cost. The preference is given to execute the request in the same SPs.

Let us represent w as the preference factor for assigning the service request. Consider w_1, w_2, and w_3 as the weights for rent-out cost, available storage, and available computation, respectively. It is very crucial to assign weights to each of the factors. This is because, the weight affects the profit for the RSU on one side and QoS for the vehicle request on the other side.

Let us discuss each of them:

- w is the preference given for node types. This factor is important for the fairness aspect. First, priority is given to current SP. This increases the resource utilization of the current SP and thus increases the profit. Then, the preference is given to other SPs of the network. In case none of the SPs satisfy requirement of the request, priority is given to PVs and CC that satisfy the QoS requirement and generate the highest profit for the current SP.

- w_1 is the weight assigned for rent-out costs charged for service requests. It is directly associated with the profit for the SP. In case of a request transfer to one of the RSUs of the same SP, the rent-out cost is considered to be zero. Here, a higher rent-out cost means lesser profit for the SP.

- w_2 is the weight assigned for the available storage in the target node. Although the requested amount of storage is necessary for all the nodes to be chosen for the assignment. However, larger storage in the targeted node helps to minimize the context switch time in case additional data is requested in the subsequent time instant.

- w_3 is the weight assigned for the available computation in the target node. Although the requested amount of computation is necessary for all the nodes to be chosen for the assignment. However, larger computation in the targeted node helps to minimize the delay.

If a request has to be assigned to one RSU of the same SP, then

$$w = 1 \tag{5.11}$$

If the request has to be assigned to one RSU of a different SP, then

$$w = 0.8 \tag{5.12}$$

If the request has to be assigned to one PV of SP, then

$$w = 0.3 \tag{5.13}$$

If the request has to be assigned to the CC (Π), then

$$w = 0.2 \tag{5.14}$$

Profit obtained by the SPs depends on the type of service and the amount of service the requested vehicle needs from the targeted node. Each service request from the vehicle may have a combination of storage, computation, and content delivery type of service. The weight assignment depends on the amount of resources the requested vehicle needs. The values of weights chosen for rent-out cost, storage, and computation as 0.5, 0.25, and 0.25, respectively. This is because the rent-out cost directly affects the profit of the SP. The storage and computation affect the QoS, delay, and number of requests completed. The weight is represented by Ψ and is defined as follows:

$$\Psi = w \times \left(w_1 \times \left(\frac{1}{P_i^s \times s_k + P_i^c \times c_k + P_i^d \times t_k} \right) + w_2 \times \frac{S_i}{s_k} + w_3 \times \frac{C_i}{c_k} \right) \tag{5.15}$$

5.6.2 Adaptive Algorithm for Profit Maximization (AAPM)

Algorithm 10: Adaptive Algorithm for Profit Maximization (AAPM)

Initialize:
1 $R \leftarrow NULL,\ P \leftarrow NULL,\ Pi \leftarrow NULL,\ F^r \leftarrow 0,\ F^p \leftarrow 0,\ F^c \leftarrow 0,\ E \leftarrow 0.0,\ I \leftarrow 0$
2 **for** $(i = 0; i < T; i++)$ **do**
3 $\quad Identify\_Nodes()$
4 $\quad Assign\_Requests()$
5 $\quad Process\_Requests()$
6 $Calculate\ F\ for\ each\ vehicles$
7 $Calculate\ E\ for\ each\ SP$

Algorithm 10 is the main algorithm that calls three different modules of the algorithm in an iterative manner from time 0 to T. At the end, it calculates the total number of requests completed processing and the total profit of each SP.

Algorithm 11: Identify_Node

```
   Initialize
1  N^c ← NULL
2  for (i = 0; i < N; i = i + 1) do
3      for (i = 0; i < Γ; i = i + 1) do
4          if ((r_i^remSt > q_i^sn)&&(r_i^remCm > q_i^cn)) then
5              Add r_i to N^c
6      for (i = 0; i < B; i = i + 1) do
7          if ((p_i^remSt > q_i^sn)&&(p_i^remCm > q_i^cn)) then
8              Add p_i to N^c
9  Add Π to N^c
10 return N^c
```

Algorithm 11 checks for the availability of sufficient storage and computing needs in RSUs and PVs of each SP. In line 4, r_i^{remSt} is the remaining storage of RSU r_i, q_i^{sn} is the storage need of the request q_i, r_i^{remCm} is the remaining computing resources of RSU r_i, and q_i^{cn} is the storage need of the request q_i. The nodes that satisfy the storage need and the computation need get added to the candidate node list N^c along with the CC node. This is because, we assume that the CC always has sufficient storage and computation. Similarly, in line 7, p_i^{remSt} is the remaining storage of PV p_i and p_i^{remCm} is the remaining computing resources of PV p_i.

Algorithm 12 takes a list of candidate nodes and a list of requests as input to assign the requests to candidate nodes. *Assignments* is a dictionary to store the assignments. Then the algorithm calculates the weight and available slots in case a request is assigned to the candidate nodes. QoS parameters are checked for each possible assignment. In case the requirement is not satisfied, the weight is considered as zero. The node with highest weight is chosen for assignment in the available time slot for each unassigned request. At the end, information related to the assignment, request list, and node list gets updated.
Algorithm 13 takes the requests one by one from the allotment list and processes them within the allocated slots. In case a request completes its processing at any given instant, the resources allocated to that request get released.

Algorithm 12: Assign_Requests

Initialize:
1 $Selected\_nodes \leftarrow N^c$
2 $Assignments \leftarrow NULL$
3 **for** *(each* q_i^u *in* Q*)* **do**
4 **for** *(each node in Selected_nodes)* **do**
5 *calculate* $weight(q_i^u)$
6 *calculate* $available - slots$
7 **if** $(QoS(q_i^u, node) == false)$ **then**
8 $weight(q_i^u) = 0.0$
9 **for** *(each* q_i^u *in* Q*)* **do**
10 $w_{max} \leftarrow Max(weight(q_i^u))$
11 $indexof(w_{max}) \leftarrow q_i^u$
12 $duration \leftarrow available - slots$
13 $Assignments = Assignments + Ass_t$
14 $Calculate\ Cost$
15 $Update\ Q$
16 $Update\ Selected\_nodes$
17 **for** *(each* r_i *in* R*)* **do**
18 $Calculate\ P_i, cost_i^{infra}, cost_i^{rent}, cost_i^{pen}, E_i$
19 **return** *Assingments*

Algorithm 13: Process_Request

Initialize:
1 $Allotment\_list \leftarrow Assignments$
2 **for** $(r_i$ *in* $RSU)$ **do**
3 **for** $(q_i$ *in* $allotment\_list)$ **do**
4 $process(q_i)$
5 **if** $(complete(q_i) == True)$ **then**
6 $release\_resource(q_i)$
7 **if** $(t > \mathcal{L}(q_i))$ **then**
8 *Append* q_i *to* I
9 **return** *Allotment_list*

Table 5.1: *Simulation parameters for AAPM*

Parameters	Values
Vehicles count	50-300
Number of time steps	890
Number of RSUs	9
Per unit storage cost of RSU	0.001$ - 0.007$
Per unit computing cost of RSU	0.001$ - 0.007$
Per unit cost of content delivery	0.002$
Per unit storage cost paid by vehicles	0.001$ - 0.005$
Per unit computing cost paid by vehicles	0.001$ - 0.005$
Per unit cost of content delivery paid by vehicles	0.0015$
Penalty per unit of storage resource paid by SPs	0.001$ - 0.009$
Penalty per unit of computing resource paid by SPs	0.001$ - 0.009$
Penalty per unit time of content delivery paid by SPs	0.0022$
RSU's transmission range	500 Meters
Total storage capacity of each RSU	6000 MB
Total computing capacity of each RSU	1000 Units

5.7 Simulation Results and Discussion

This section presents the results achieved through the implementation of AAPM using a Java-based discrete event simulator. The study focuses on a lower Manhattan city scenario [60] encompassing a 10-square-kilometers area with bidirectional roads. Nine RSUs are uniformly distributed within this scenario, each having a transmission range of 500 meters. Consequently, the RSUs do not overlap, leaving certain areas without RSU coverage. The movement of vehicles in the simulation is generated using the Simulation of Urban MObility (SUMO) [18]. The simulation parameter is specified in the table 5.1. Important simulation parameters are taken from the existing works [22, 27].

We have evaluated our proposed algorithm for periodic applications. By periodic application, we mean an application that generates data periodically. We have compared the result of AAPM with Simulated annealing-based Migrating Birds Optimization (SMBO) [64] and Parked Vehicle-assisted Task Offloading (PVATO) [65]. Along with that, we compare the results with a GREEDY approach. By GREEDY approach we mean that a request is assigned to RSUs of the same SP first and if sufficient resources are not available it is

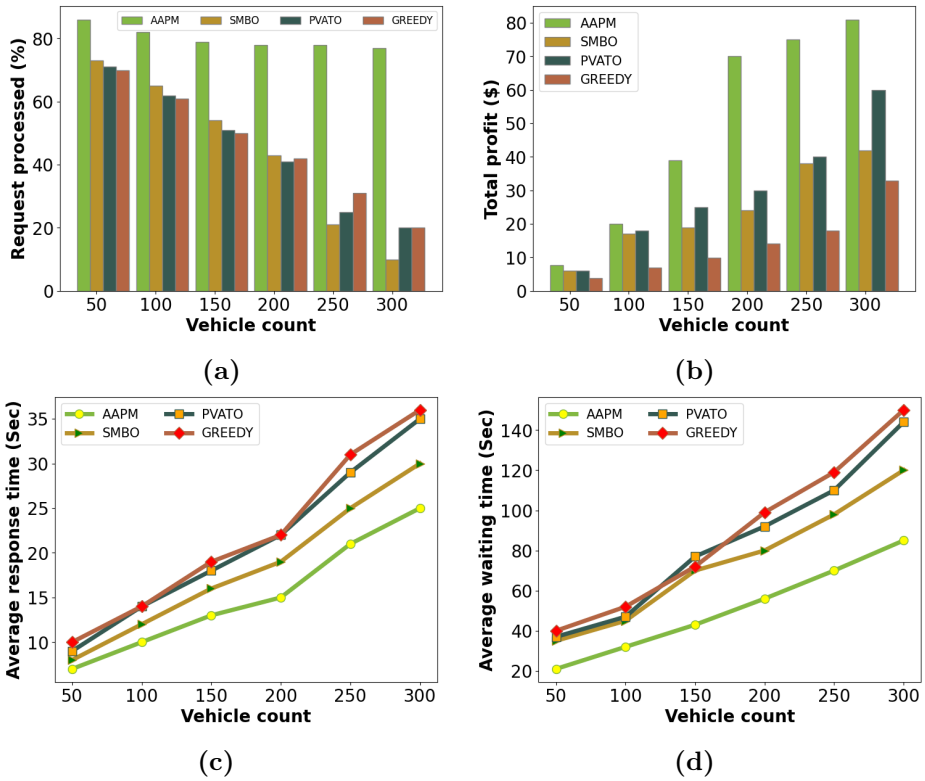


Figure 5.3: *Performance of AAPM with variation of vehicle count*

assigned to RSUs of other SPs. However, AAPM assigns the request to one of the target nodes. The node may be an RSU, PV, or CC.

In Figure 5.3 performance of AAPM is shown with varying numbers of vehicles. The vehicle count is varied from 50 to 300 to evaluate the algorithm. Figure 5.3a shows the request completion percentage for different vehicle counts. It can be observed that AAPM outperforms other algorithms because of the consideration of remaining storage and computation of available nodes. AAPM allocates the request to the node which can complete the request with a minimum time and gives less preference to PVs and CC. This reduces the delay and increases the chances of processing a larger number of requests. Figure 5.3b demonstrates the result for total profit obtained by SPs. AAPM shows an improved performance as compared to the existing algorithms as well as the GREEDY algorithm. This is because of consideration of rent-out cost and selection of right node types before allocating a request. The trends of average response time and average waiting time are shown in Figure 5.3c and Figure 5.3d respectively. AAPM outperforms other algorithms for these metrics.

This is because of the availability of multiple options for a request during the assignment phase. Based on the resource requirements of the request, a suitable node is chosen for the assignment that minimizes the response time and waiting time of the resource requests.

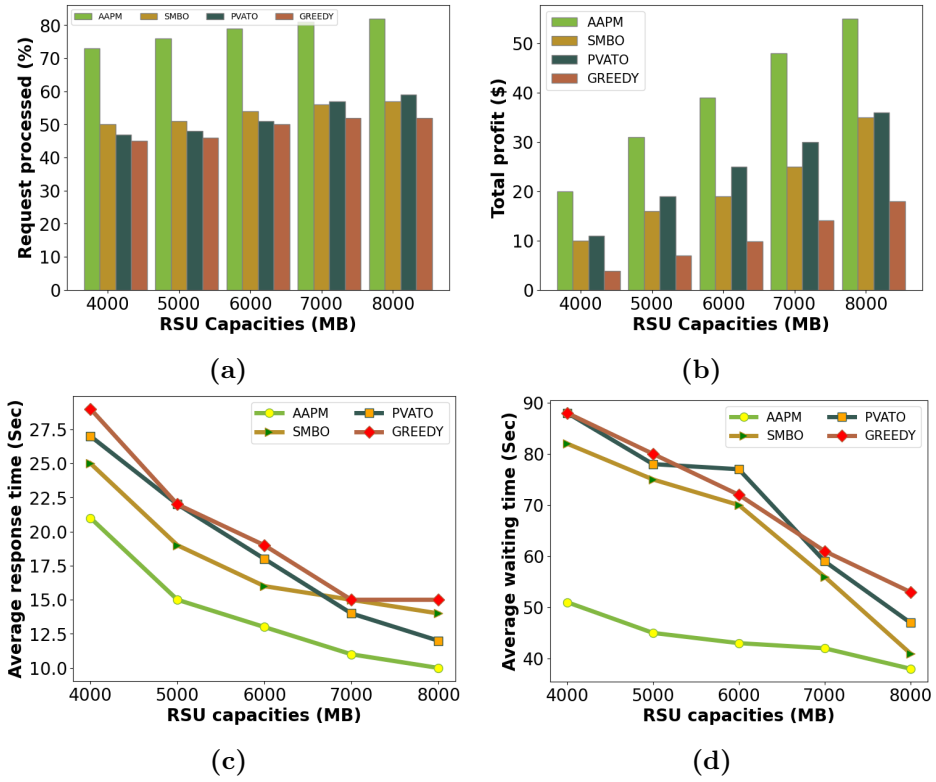


Figure 5.4: *Performance of AAPM with variation of RSU capacity*

Figure 5.4 shows the performance of AAPM with variation in RSU capacities. Value of the vehicle count is taken as 150 in this scenario. Figure 5.4a shows the performance of AAPM for request completion percentage with varying RSU capacities. It can be seen that AAPM performs better as compared to other existing algorithms for different RSU capacities. The reason behind this is the use of preference values while assigning a request to one of the node types. An increase in RSU capacities leads to more number request processing at the SP, which leads to more request completion. Figure 5.4b shows the change in profit values with respect to the change in RSU capacities. This is because of efficient resource utilization of SP in the case of AAPM which increases the profit of SPs. Figure 5.4c and Figure 5.4d show the response time and waiting time respectively. AAPM outperforms other algorithms for these scenarios. This is because of the availability of multiple options

for SPs in AAPM while there is a need for request assignments. The resources available to RSUs belonging to other SPs are used while there is a need.

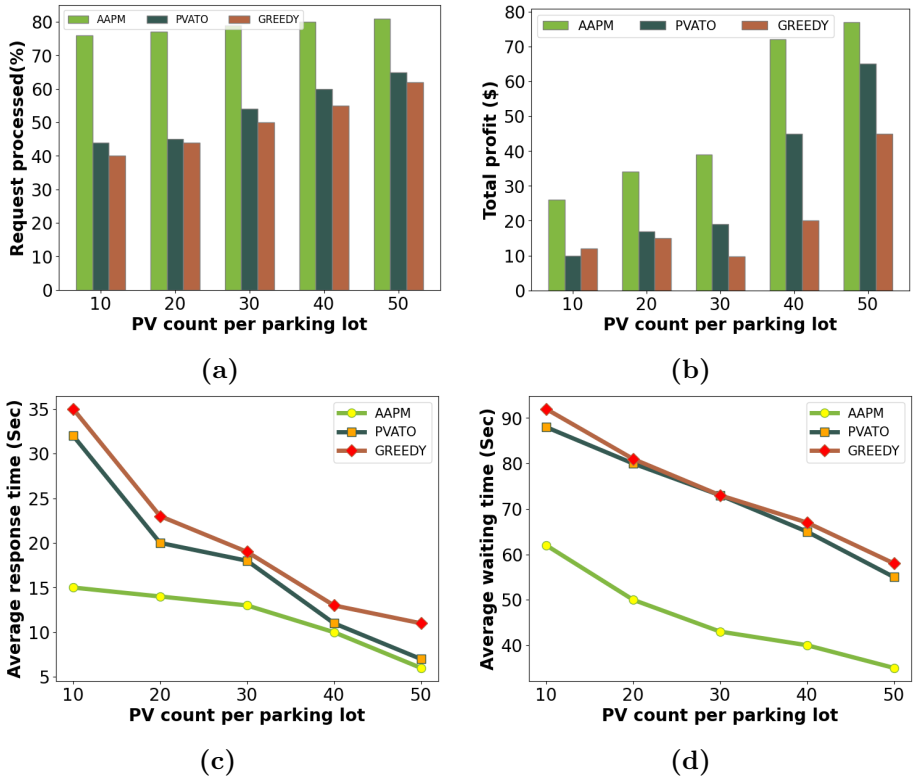


Figure 5.5: *Performance of AAPM with variation of number of parked vehicles*

Figure 5.5 shows the performance of AAPM with the variation of PV count at each parking lot. In this scenario, we have not considered SMBO as it does not use PVs. It can be observed in Figure 5.5a that the request completion percentage is higher in the case of AAPM as compared to PVATO and GREEDY approaches. This is because, the overloaded RSU gives more preference to the PVs. Figure 5.5b shows the total profit incurred by SPs. The increase in PVs leads to the use of low-cost resources by the SPs which leads to a decrease in cost. Thus, the profit increases for the SP. Figure 5.5c and Figure 5.5d show the variation of response time and waiting time of the algorithm respectively. In this scenario also AAPM performs better than the existing algorithms. This is because, it uses the resources of PVs and the CC while processing data. The availability of various types of resources help the scheduler to choose the right resources while scheduling.

5.8 Chapter Summary

In this chapter, the proposed algorithm AAPM maximizes the profit of SPs in heterogeneous vehicular environments without affecting the QoS. To increase resource availability, PVs, CC, and RSUs are considered as options while doing request assignments. An SP consists of a fixed number of RSUs which interact with the vehicles and other RSUs of the network. AAPM improves request processing by 46.30% compared to SMBO, and by 54.90% compared to PVATO. The proposed algorithm improves total profit by 62.50% compared to SMBO and by 30% compared to PVATO. The average response time is reduced by 18.75% compared to SMBO and by 27.78% compared to PVATO. AAPM reduces average waiting time by 38.57% compared to SMBO and by 44.16% compared to PVATO.

6

Conclusions and Future Prospectives

In this chapter we summarize the work done, highlight the contributions, and suggest the directions for possible future work.

6.1 Conclusions

In this thesis, we have proposed a set of algorithms for request assignment at RSU that helps in load balancing. To maximize the number of requests processed and minimize the end-to-delay, we have used three different approaches. In the first approach, we have utilized the resources of all the RSUs in the network. Here, we have used VM migrations from the overloaded RSU to other RSUs with available resources and focused on minimizing the cost of migration. In the second approach, we have rented the resources from other RSU's of the network for that we have proposed a price model. Here, we have tried to maximize the number of requests served and minimize the total cost. The total cost consists of migration cost and rent-out cost. In the third approach, we have increased the availability of resources by renting the required resources from CC, PVs and other RSUs. Here, we have considered SPs which consist of more than one RSUs. To maximize the profit of SPs without affecting QoS, we have proposed an algorithm that assigns the request to one of the nodes out of PVs, CC and RSUs. In particular, we have proposed the following algorithms:

1. **AALB:** An Application Aware Load Balancing algorithm for RSU in VANETs. This algorithm considers application types, deadline of the request along with the storage and computation availability of RSUs. This algorithm maximizes the number of requests processed and minimizes the migration cost.

2. **ERR:** Resource Renting Algorithm for Load Balancing in Road Side Units. This algorithm focuses on renting the resources from other RSUs of the network. It maximizes the number of requests processed and minimizes the total cost. Total cost consists of migration costs and rent-out costs.

3. **AAPM:** Adaptive Algorithm for Profit Maximization in Heterogeneous Vehicular Environment. This algorithm takes care of the scenarios where none of the RSUs in the network is able to process the request. To increase the availability of the resources, other nodes with available resources are considered. Those nodes are CC, PVs and RSUs of another SPs. This algorithm maximizes the profit of SPs without affecting the QoS.

Here, we have shown a basic comparison of the proposed algorithms with the existing algorithms for the 150-vehicle scenario.

Table 6.1: *Performance comparison of load balancing algorithms*

Metrics	**AALB**	**ERR**	**AAPM**	**JSCO**	**MAMTS**	**DCORA**	**SMBO**	**PVATO**
VM Completion (%)	**82**	**89**	–	53	60	35	–	–
Migration Cost ($)	**50**	–	–	800	–	–	–	–
Total Cost ($)	–	**25**	–	125	–	–	–	–
Average Delay(sec)	162	**151**	–	400	–	–	–	–
Requests Processed (%)	–	–	**79**	–	–	–	54	51
Total Profit ($)	–	–	**39**	–	–	–	24	30
Avg. Res. Time(sec)	–	–	**13**	–	–	–	16	18
Avg. Wait. Time(sec)	–	–	**43**	–	–	–	70	77

6.2 Future Prospectives

The work reported in the chapters of this thesis provides ample scope and shows several clear directions for future research endeavors. One can think of designing prediction-based

resource reservation policy before the actual scheduling. This can enhance the scheduling decisions for more optimal task assignments. Task dependencies are problems in the case of task partition. This can be taken into consideration while making the assignment decisions. The number of applications generating resource requests in a vehicle can be considered as a factor while formulating the problem in case of task assignment in VANETs. This is because with increase in the number of applications and increased data generation rate of the application may lead to the generation of simultaneous requests from a single vehicle. This factor can be taken into consideration while doing the task assignment and VM migration. The nearby vehicles with available resources can take part in load balancing either individually or by forming a cloud. Those resources are vital from a load-balancing point of view. This can be considered in future research directions to make the assessment decision better. However, the problem of quick disconnection and limited transmission range of vehicles is always a problem in the case of fast-moving vehicles in the case of VANETs.

The proposed load balancing framework for VANETs presents multiple avenues for future research that could enhance its effectiveness and applicability in real-world urban environments. One of the most promising directions is the integration of 5G and beyond. The ultra-low latency, high reliability, and network slicing capabilities of 5G can be leveraged to dynamically prioritize traffic loads across roadside units (RSUs), parked vehicles (PVs), and central cloud infrastructure. This can support diverse quality-of-service (QoS) requirements, particularly for critical applications like emergency services and autonomous vehicle coordination.

Expanding the architecture to include edge and fog computing is another valuable direction. Future work can design layered coordination strategies between RSUs, PVs, and fog nodes to enhance real-time data processing. Additionally, incorporating energy-aware scheduling and federated learning at the edge can improve scalability while preserving data privacy and minimizing energy consumption, especially for electric vehicles acting as edge nodes. Incorporating AI-driven predictive models can enable proactive load balancing. Machine learning models trained on historical traffic, weather, and event data can anticipate surges in demand, while reinforcement learning can adapt load distribution strategies in real time based on continuous feedback. Together, these future research avenues can significantly enhance the resilience, efficiency, and adaptability of VANET-based load balancing systems, contributing meaningfully to the development of next-generation smart cities.

Following are the three major research directions that can be taken into consideration in the future:

- In Chapter 3, applications running in vehicles generate data based on the application types and user's preferences. Prediction of the amount of data generation in the future is suitable for the assignment of VMs to RSUs. Lack of information about future data makes some assignments which lead to unnecessary VM migration. Thus, there is an increase in migration cost and end-to-end delay. In this scenario, a prediction of request assignment is a suitable option. However, existing prediction-based algorithms require a huge amount of computation which may not be able to run on those RSUs that are overloaded with the current requests. Thus, there is a need for a suitable approach to deal with this scenario.

- In Chapter 4, the cost of deployment of RSU and its maintenance cost makes it difficult for the authorities to cover an entire city. This is because the RSU setup and maintenance cost is too high RSU [66]. However, in an overloaded scenario, the placement of mobile RSUs is one of the options that alleviates the the chance of overloading and service disruption. It is critical to decide how many mobile RSUs are needed for the overloaded scenario along with their position of deployment.

- In Chapter 5, to increase the availability of resources, PVs and CC are considered. However, the resources of nearby vehicles are not considered. The nearby vehicles may have available resources that can be used to eliminate overloaded scenarios in RSU. The nearby vehicles might be taken into consideration individually or in a group. In case, those vehicles are considered individually, the available resources may not be sufficient to serve the request and in case of cluster how to decide the cluster size in case of fast-moving vehicles with different speeds and directions. All these problems have to be taken care while using the resources of nearby vehicles.

Disseminations out of the Thesis Work

Journals

1. **Swagat Ranjan Sahoo**, Moumita Patra, and Arobinda Gupta, "**AALB: Application Aware Load Balancing Algorithm for Road Side Units", Elsevier Journal on Vehicular Communications**, vol. 36, pp. 100475, Aug. 2022. [Contribution 1]

2. **Swagat Ranjan Sahoo**, Moumita Patra, Shivank Thapa, and Arobinda Gupta, "**Resource Renting Algorithm for Load Balancing in Road Side Units**", submitted to Elsevier Journal of Adhoc Networks, May. 2025. [Contribution 3]

3. **Swagat Ranjan Sahoo** and Moumita Patra, "**AAPM: Adaptive Algorithm for Profit Maximization in Heterogeneous Vehicular Environment**", submitted to Springer Journal of Wireless Networks, Aug. 2024 [Contribution 4]

Conferences

1. **Swagat Ranjan Sahoo**, Moumita Patra, and Arobinda Gupta, "**MDLB: A Matching based Dynamic Load Balancing Algorithm for Road Side Units**", in Proceedings of IEEE International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 291-296, Aug. 2021. [Contribution 1]

2. Shivank Thapa, **Swagat Ranjan Sahoo**, Moumita Patra, and Arobinda Gupta, "**A Novel Cost-Aware Load Balancing Algorithm for Road Side Units in Internet of Vehicles**", in Proceedings of 18th IEEE International Conference on Network and Service Management (CNSM), pp. 359-363, Dec. 2022. [Contribution 2]

References

[1] F. Cunha, L. Villas, A. Boukerche, G. Maia, A. Viana, R. A. Mini, and A. A. Loureiro, "Data communication in VANETs: Protocols, applications and challenges," *Elsevier Ad hoc networks*, vol. 44, pp. 90–103, Jul. 2016. [Pg.1]

[2] E. C. Eze, S.-J. Zhang, E.-J. Liu, and J. C. Eze, "Advances in vehicular ad-hoc networks (VANETs): Challenges and road-map for future development," *Springer International Journal of Automation and Computing*, vol. 13, pp. 1–18, Jan. 2016. [Pg.1]

[3] N. Moustafa, M. Patra, and V. Tamarapalli, "Cost-and-delay aware dynamic resource allocation in federated vehicular clouds," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 6159–6171, May. 2021. [Pg.3], [Pg.16]

[4] C. Maag, D. Muhlbacher, C. Mark, and H.-P. Kruger, "Studying effects of Advanced Driver Assistance Systems (ADAS) on individual and group level using multi-driver simulation," *IEEE Intelligent Transportation Systems Magazine*, vol. 4, no. 3, pp. 45–54, Aug. 2012. [Pg.3]

[5] J. Wright, J. Garrett, C. Hill, G. Krueger, J. Evans, S. Andrews, C. Wilson, R. Rajbhandari, and B. Burkhard, "American association of state highway and transportation officials. national connected vehicle field infrastructure footprint analysis," Technical Report FHWA-JPO-14-125, Tech. Rep., Jun. 2014. [Pg.3]

[6] M. Masdari, S. S. Nabavi, and V. Ahmadi, "An overview of virtual machine placement schemes in cloud computing," *Elsevier Journal of Network and Computer Applications*, vol. 66, pp. 106–127, May. 2016. [Pg.4]

[7] R. Yu, Y. Zhang, H. Wu, P. Chatzimisios, and S. Xie, "Virtual machine live migration for pervasive services in cloud-assisted vehicular networks," in *IEEE International*

Conference on Communications and Networking in China (CHINACOM), Aug. 2013, pp. 540–545. [Pg.4]

[8] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang, "Toward cloud-based vehicular networks with efficient resource management," *IEEE Network*, vol. 27, no. 5, pp. 48–55, Sep. 2013. [Pg.4], [Pg.21]

[9] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, Oct. 2015. [Pg.5]

[10] J. Feng, Z. Liu, C. Wu, and Y. Ji, "AVE: Autonomous vehicular edge computing framework with aco-based scheduling," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 12, pp. 10 660–10 675, Jun. 2017. [Pg.5]

[11] X.-Q. Pham, T. Huynh-The, E.-N. Huh, and D.-S. Kim, "Partial computation offloading in parked vehicle-assisted multi-access edge computing: A game-theoretic approach," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 9, pp. 10 220–10 225, Jun. 2022. [Pg.5]

[12] X.-Q. Pham, T.-D. Nguyen, V. Nguyen, and E.-N. Huh, "Joint node selection and resource allocation for task offloading in scalable vehicle-assisted multi-access edge computing," *Symmetry*, vol. 11, no. 1, Jan. 2019. [Pg.5]

[13] Z. Zhou, P. Liu, J. Feng, Y. Zhang, S. Mumtaz, and J. Rodriguez, "Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3113–3125, Jan. 2019. [Pg.5]

[14] X. Huang, R. Yu, J. Liu, and L. Shu, "Parked vehicle edge computing: Exploiting opportunistic resources for distributed mobile applications," *IEEE Access*, vol. 6, pp. 66 649–66 663, Nov. 2018. [Pg.5]

[15] F. H. Rahman, A. Y. M. Iqbal, S. S. Newaz, A. T. Wan, and M. S. Ahsan, "Street parked vehicles based vehicular fog computing: TCP throughput evaluation and future research direction," in *IEEE International Conference on Advanced Communication Technology (ICACT)*, May. 2019, pp. 26–31. [Pg.5]

[16] X. Huang, R. Yu, D. Ye, L. Shu, and S. Xie, "Efficient workload allocation and user-centric utility maximization for task scheduling in collaborative vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3773–3787, Mar. 2021. [Pg.5]

[17] K. Wang, H. Yin, W. Quan, and G. Min, "Enabling collaborative edge computing for software defined vehicular networks," *IEEE Network*, vol. 32, no. 5, pp. 112–117, Mar. 2018. [Pg.5]

[18] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic traffic simulation using SUMO," in *IEEE International Conference on Intelligent Transportation Systems*, Dec. 2018, pp. 2575–2582. [Pg.7], [Pg.42], [Pg.67], [Pg.101]

[19] F. Goudarzi, H. Asgari, and H. S. Al-Raweshidy, "Traffic-aware VANET routing for city environments—a protocol based on ant colony optimization," *IEEE Systems Journal*, vol. 13, no. 1, pp. 571–581, Mar. 2019. [Pg.13]

[20] J. Shi, Z. Yang, H. Xu, M. Chen, and B. Champagne, "Dynamic resource allocation for lte-based vehicle-to-infrastructure networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5017–5030, Mar. 2019. [Pg.13]

[21] K. Abrougui, A. Boukerche, and H. Ramadan, "Efficient load balancing and QoS-based location aware service discovery protocol for vehicular ad hoc networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2012, pp. 1–15, Mar. 2012. [Pg.13]

[22] G. G. M. N. Ali, E. Chan, and W. Li, "On scheduling data access with cooperative load balancing in vehicular ad hoc networks," *Springer The Journal of Supercomputing*, vol. 67, no. 2, pp. 438–468, Feb. 2014. [Pg.14], [Pg.19], [Pg.42], [Pg.67], [Pg.68], [Pg.101]

[23] G. Li, Y. Yao, J. Wu, X. Liu, X. Sheng, and Q. Lin, "A new load balancing strategy by task allocation in edge computing based on intermediary nodes," *Springer EURASIP Journal on Wireless Communications and Networking*, vol. 1, pp. 1–10, Jan. 2020. [Pg.14]

[24] H. T. Hashemi and S. Khorsandi, "Load balanced VANET routing in city environments," in *IEEE Vehicular Technology Conference*, May. 2012, pp. 1–6. [Pg.14]

[25] H. Chi-Fu and J.-H. Jhang, "Efficient RSU selection approaches for load balancing in vehicular ad-hoc networks," *TAETI, Advances in Technology Innovation*, vol. 5, no. 1, pp. 56–63, Jan. 2019. [Pg.15]

[26] C. Lin, D. Deng, and C. Yao, "Resource allocation in vehicular cloud computing systems with heterogeneous vehicles and road side units," *IEEE IoT Journal*, vol. 5, no. 5, pp. 3692–3700, Oct. 2018. [Pg.15], [Pg.19]

[27] L. Li, H. Zhou, S. X. Xiong, J. Yang, and Y. Mao, "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. [Pg.15], [Pg.19], [Pg.42], [Pg.67], [Pg.68], [Pg.101]

[28] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2092–2104, Dec. 2019. [Pg.15]

[29] K. Zhang, Y. Mao, S. Leng, A. Vinel, and Y. Zhang, "Delay constrained offloading for mobile edge computing in cloud-enabled vehicular networks," in *IEEE International Workshop on Resilient Networks Design and Modeling*, Oct. 2016, pp. 288–294. [Pg.15], [Pg.19]

[30] 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, Oct. 2018. [Pg.15], [Pg.43]

[31] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, Jun. 2019. [Pg.16], [Pg.19], [Pg.43]

[32] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, and F. Yang, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4961–4971, Feb. 2020. [Pg.16], [Pg.19], [Pg.43]

[33] L. Chen, X. Li, and R. Ruiz, "Resource renting for periodical cloud workflow applications," *IEEE Transactions on Services Computing*, vol. 13, no. 1, pp. 130–143, Mar. 2017. [Pg.16]

[34] B. K. Ray, A. Saha, S. Khatua, and S. Roy, "Toward maximization of profit and quality of cloud federation: solution to cloud federation formation problem," *Springer The Journal of Supercomputing*, vol. 75, no. 2, pp. 885–929, Feb. 2019. [Pg.16]

[35] H. Deng, L. Huang, H. Xu, X. Liu, P. Wang, and X. Fang, "Revenue maximization for dynamic expansion of geo-distributed cloud data centers," *IEEE Transactions on Cloud Computing*, vol. 8, no. 3, pp. 899–913, Sep. 2020. [Pg.16]

[36] J. Mei, K. Li, Z. Tong, Q. Li, and K. Li, "Profit maximization for cloud brokers in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 190–203, Jan. 2019. [Pg.17]

[37] A. Zhou, Q. Sun, L. Sun, J. Li, and F. Yang, "Maximizing the profits of cloud service providers via dynamic virtual resource renting approach," *EURASIP Journal on Wireless Communications and Networking*, vol. 2015, no. 1, pp. 1–12, Mar. 2015. [Pg.17]

[38] B. Brik, N. Lagraa, N. Tamani, A. Lakas, and Y. Ghamri-Doudane, "Renting out cloud services in mobile vehicular cloud," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 10, pp. 9882–9895, Jul. 2018. [Pg.17]

[39] J. Karjee, P. Naik S, K. Anand, and V. N. Bhargav, "Split computing: Dnn inference partition with load balancing in iot-edge platform for beyond 5g," *Measurement: Sensors*, vol. 23, p. 100409, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2665917422000435 [Pg.18]

[40] S. A. Lahham, D. Wu, E. Hossain, X. Liu, and G. Dudek, "Probabilistic mobility load balancing for multi-band 5g and beyond networks," in *2024 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2024, pp. 1673–1678. [Pg.18]

[41] G. Li, Y. Yao, J. Wu, X. Liu, X. Sheng, and Q. Lin, "A new load balancing strategy by task allocation in edge computing based on intermediary nodes," *Springer EURASIP*

Journal on Wireless Communications and Networking, vol. 2020, no. 1, p. 3, Jan. 2020. [Pg.19]

[42] H. T. Hashemi and S. Khorsandi, "Load balanced vanet routing in city environments," in *2012 IEEE 75th Vehicular Technology Conference (VTC Spring)*. IEEE, 2012, pp. 1–6. [Pg.19]

[43] C.-F. Huang and J.-H. Jhang, "Efficient rsu selection approaches for load balancing in vehicular ad hoc networks," *Adv. Technol. Innov*, vol. 5, no. 1, pp. 56–63, 2020. [Pg.19]

[44] 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. [Pg.19], [Pg.68]

[45] Y. Wu, J. Wu, L. Chen, J. Yan, and Y. Han, "Load balance guaranteed vehicle-to-vehicle computation offloading for min-max fairness in vanets," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 11 994–12 013, Aug. 2022. [Pg.20]

[46] D. L. Msongaleli and K. Kucuk, "Optimal resource utilisation algorithm for visible light communication-based vehicular ad-hoc networks," *IET Intelligent Transport Systems*, vol. 14, no. 2, pp. 65–72, Feb. 2020. [Pg.20]

[47] T.-Y. Wu, M. S. Obaidat, and H.-L. Chan, "Qualityscan scheme for load balancing efficiency in vehicular ad hoc networks," *Journal of Systems and Software*, vol. 104, pp. 60–68, Jun. 2015. [Pg.21]

[48] K. Kamini and R. Kumar, "VANET parameters and applications: A review," *Global Journal of Computer Science and Technology,*, vol. 10, no. 7, pp. 1–6, Jan. 2010. [Pg.21]

[49] M. Lee and T. Atkison, "VANET applications: Past, present, and future," *Vehicular Communications*, vol. 28, p. 100310, Apr. 2021. [Pg.21], [Pg.44]

[50] M. S. Akbar, M. S. Khan, K. A. Khaliq, A. Qayyum, and M. Yousaf, "Evaluation of IEEE 802.11 n for multimedia application in VANET," *Procedia Computer Science*, vol. 32, pp. 953–958, Jun. 2014. [Pg.21], [Pg.43]

[51] E. Lee, E.-K. Lee, M. Gerla, and S. Y. Oh, "Vehicular cloud networking: architecture and design principles," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 148–155, Feb. 2014. [Pg.21]

[52] C. Tang, C. Zhu, X. Wei, H. Wu, Q. Li, and J. J. Rodrigues, "Intelligent resource allocation for utility optimization in rsu-empowered vehicular network," *IEEE access*, vol. 8, pp. 94 453–94 462, 2020. [Pg.22]

[53] J. B. Kenney, "Dedicated short-range communications (DSRC) standards in the united states," *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1162–1182, Jul. 2011. [Pg.22]

[54] Y. Zeng, X. Wu, and J. Cao, "Research and implementation of hungarian method based on the structure index reduction for dae systems," *Journal of Algorithm and Comput Tech*, vol. 8, no. 2, pp. 219–231, Jun. 2014. [Pg.32], [Pg.63]

[55] H. Cui, J. Zhang, C. Cui, and Q. Chen, "Solving large-scale assignment problems by kuhn-munkres algorithm," in *2nd International Conference on Advances in Mechanical Engineering and Industrial Informatics(AMEII 2016)*, Jan. 2016. [Pg.32], [Pg.63]

[56] S. Buzzi, C. D'Andrea, M. Fresia, Y.-P. Zhang, and S. Feng, "Pilot assignment in cell-free massive MIMO based on the Hungarian algorithm," *IEEE Wireless Communications Letters*, vol. 10, no. 1, pp. 34–37, Aug. 2020. [Pg.32]

[57] H. Ye, Y. Chen, and X. Li, "Resource pricing model based on double auction for the cloudlet federation," *ICIC International Journal of Innovative Computing, Information and Control*, vol. 17, no. 1, pp. 431–446, Apr. 2023. [Pg.55]

[58] N. V. Sahinidis, "Mixed-integer nonlinear programming 2018," *Springer Optimization and Engineering*, vol. 20, no. 2, pp. 301–306, Apr. 2019. [Pg.59]

[59] Y. Tang, N. Cheng, W. Wu, M. Wang, Y. Dai, and X. Shen, "Delay-minimization routing for heterogeneous vanets with machine learning based mobility prediction," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3967–3979, Apr. 2019. [Pg.59]

[60] H. Noori, "Impact of VANET-based traffic signal control on the response time of emergency vehicles in realistic large scale urban area," in *IEEE International Conference on Communication Workshop*, Jun. 2013. [Pg.67], [Pg.101]

[61] S. R. Sahoo, M. Patra, and A. Gupta, "AALB: Application-aware load balancing algorithm for road side units," *Elsevier Vehicular Communications*, vol. 36, p. 100475, Aug. 2022. [Pg.68]

[62] J. I. Mbegbu and U. V. Echebiri, "Juchez probability distribution: properties and applications," *Asian Journal of Probability and Statistics*, vol. 20, no. 2, pp. 56–71, Nov. 2022. [Pg.74]

[63] K. Dorgham, I. Nouaouri, H. Ben-Romdhane, and S. Krichen, "A hybrid simulated annealing approach for the patient bed assignment problem," *Procedia Computer Science*, vol. 159, pp. 408–417, Oct. 2019. [Pg.94]

[64] H. Yuan and M. Zhou, "Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 1277–1287, Jul. 2021. [Pg.101]

[65] C. Ma, J. Zhu, M. Liu, H. Zhao, N. Liu, and X. Zou, "Parking edge computing: Parked-vehicle-assisted task offloading for urban VANETs," *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 9344–9358, Jun. 2021. [Pg.101]

[66] A. Guerna, S. Bitam, and C. T. Calafate, "Roadside unit deployment in internet of vehicles systems: A survey," *MDPI Sensors*, vol. 22, no. 9, p. 3190, Apr. 2022. [Pg.109]

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati 781039, India