WORKLOAD ORCHESTRATION FOR MULTI-TIER MULTI-ACCESS EDGE COMPUTING SYSTEMS

by

Çağatay Sönmez

B.S., Computer Engineering, Dokuz Eylül University, 2008

M.S., Computer Engineering, Boğaziçi University, 2012

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2020

ACKNOWLEDGEMENTS

Firstly, I would like to sincerely thank my thesis supervisor Prof. Cem Ersoy. His guidance has always been with me to complete my Ph.D. journey. I always admire his courage, vision, and wisdom, which will continue inspiring me in my academic career. His positive thinking and humor also helped a lot to handle the problems I encountered during my research.

I would like to thank my thesis co-supervisor Assist. Prof. Atay Özgövde. I have always appreciated his perfection. I am grateful for his valuable advice and constructive criticisms. He pushed me to explore new directions in science and encouraged me to launch our open-source edge computing simulator for education and research.

I would like to express my deepest gratitude to my core jury members Prof. Can Özturan and Prof. Mutlu Koca for their time, constructive comments, and sincere suggestions. I would also like to thank members of the CMPE family who attend my thesis progress presentation for their insightful suggestions to make this thesis better.

I would like to thank all my friends in Netlab for their generous help and inspiration. I especially thank Can Tunca and Cihat Baktır for their contributions and warm friendships. Without Can's precious help on the Machine Learning domain, writing the last chapter of this thesis would be very painful for me.

Finally, I am really thankful to my mother Arife, and my father Mehmet for encouraging me to pursue an academic career. It would be very difficult to complete this thesis without their unquestioning support throughout my academic career.

This thesis has been supported by the Turkish Directorate of Strategy and Budget under the TAM Project number 2007K12-873.

ABSTRACT

WORKLOAD ORCHESTRATION FOR MULTI-TIER MULTI-ACCESS EDGE COMPUTING SYSTEMS

Edge computing is a broad concept covering a variety of computing technologies such as cloudlet-based computing, Fog Computing, Mobile Cloud Computing (MCC), and Multi-Access Edge Computing (MEC). All these computing paradigms have a common approach of bringing the cloud-computing capabilities to the edge of the network. The term edge used in this area refers to a location close to the end-users. Computation (task) offloading is an essential feature of edge computing. It reduces the battery consumption and makes it possible to execute applications that are unable to be executed on mobile devices due to their insufficient processing power, memory, and unpredictable network connectivity. In this regard, edge computing can enable new applications and use cases for the environments where there are many mobile users, such as university campuses, airports, and smart roads. The increasing number of smart devices that are connected to the Internet brings new challenges in terms of high communication delay and computational resources congestion. Although edge computing overcomes these challenges by bringing cloud computing capabilities to close proximity of the end-user, it presents a very dynamic and flexible environment where both computational and networking resources are utilized in real-time in accordance with the requirements. Hence, efficiently managing and orchestrating different types of resources become crucial issues. To overcome these challenges, we introduced novel edge orchestrators. Firstly, we developed an edge computing simulator, namely EgeCloudSim, to evaluate the performance of the proposed orchestration algorithms. Secondly, we proposed a fuzzy logic-based workload orchestrator for multi-tier mobile edge computing systems. As a final contribution, we presented a machine learning (ML) based workload orchestrator for multi-tier multi-access vehicular edge computing (VEC) environments.

ÖZET

ÇOK KATMANLI ÇOKLU ERİŞİMLİ KENAR BİLİŞİM SİSTEMLERİ İÇİN YÜK ORKESTRASYONU

Kenar bilişim bulutçuk tabanlı hesaplama, sis hesaplama, mobil bulut bilişim ve çoklu erişim kenar bilişim gibi çeşitli hesaplamalı teknolojileri kapsayan geniş bir kavramdır. Tüm bu bilişim paradigmaları, bulut bilişim yeteneklerini ağın kenarına getirmek için ortak bir yaklaşıma sahiptir. Burada kullanılan kenar terimi, çoğunlukla hareketli olan son kullanıcılara yakın bir konumu ifade eder. Hesaplama (görev) aktarma yöntemi, kenar bilişimin önemli bir özelliğidir. Bu yöntem mobil cihazlarda pil tüketimini azaltır ve yetersiz işlem gücü, bellek ve öngörülemeyen ağ bağlantısı nedenleriyle desteklenemeyen uygulamaları çalıştırmayı mümkün kılar. Bu bağlamda, kenar bilişim, üniversite kampüsleri, havaalanları ve akıllı yollar gibi fazla sayıda mobil kullanıcının bulunduğu ortamlar için yeni uygulamalara ve senaryolara olanak verir. İletişimdeki gecikmenin yüksek ve hesaplama kaynaklarının kısıtlı olduğu dikkate alındığında, internete bağlı akıllı cihazların sayısının artması yeni zorluklar getirmektedir. Kenar bilişim bulut bilişim yeteneklerini son kullanıcıya yakınlaştırarak bu zorlukların üstesinden gelmekle birlikte, hem hesaplama hem de ağ kaynaklarının gereksinimlere göre gerçek zamanlı olarak kullanıldığı çok dinamik ve esnek bir ortam sunmaktadır. Bu nedenle, farklı kaynak türlerini verimli bir şekilde yönetmek çok önemli bir konudur. Bu zorlukların üstesinden gelmek için yeni kenar orkestratörleri üzerinde çalıştık. İlk olarak, önerilen orkestrasyon algoritmalarının başarımını değerlendirebilmek için EgeCloudSim adlı bir kenar bilişim simülatörü geliştirdik. Sonrasında, çok katmanlı mobil kenar hesaplama sistemleri için bulanık mantık tabanlı bir yük orkestratörü önerdik. Bu katkılara ek olarak olarak, çok katmanlı çoklu erişimli araçsal kenar bilişim ortamları için makine öğrenme tabanlı bir yük orkestratörü sunduk.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . ix
LIST OF TABLES . . . xiii
LIST OF SYMBOLS . . . xiv
LIST OF ACRONYMS/ABBREVIATIONS . . . xvii
1. INTRODUCTION . . . 1
1.1. Thesis Contributions . . . 3
1.2. Thesis Outline . . . 6
2. BACKGROUND . . . 7
2.1. Edge Computing . . . 7
2.1.1. Challenges of Edge Computing . . . 10
2.2. Vehicular Edge Computing . . . 11
2.2.1. Challenges of Vehicular Edge Computing . . . 13
2.3. Modeling the Edge Computing Systems . . . 14
2.3.1. Challenges of Modeling Edge Computing Systems . . . 15
3. LITERATURE REVIEW . . . 18
3.1. Related Work on Edge Computing Simulators . . . 18
3.2. Related Work on Workload Orchestration for Edge Computing . . . 22
3.3. Related Work on Workload Orchestration for Vehicular Edge Computing 24
4. EDGECLOUDSIM . . . 29
4.1. Challenges of Developing Edge Computing Simulator . . . 32
4.2. EdgeCloudSim Architecture . . . 33
4.2.1. EdgeCloudSim Modules . . . 34
4.2.2. Ease of Use and Configurability . . . 38
4.2.3. Extensibility . . . 39
4.2.4. Complexity Evaluation . . . 40

4.3. EdgeCloudSim in Operation 42
4.3.1. Step 1: Modeling the Applications 43
4.3.2. Step 2: Modeling the Mobility 44
4.3.3. Step 3: Modeling the Network Delay 45
4.3.4. Step 4: Modeling the Edge Computing Architectures 46
4.4. Simulation Results 48
4.5. Discussion 53
5. FUZZY WORKLOAD ORCHESTRATION FOR EDGE COMPUTING . . 54
5.1. Multi-Tier Edge Computing Architecture 57
5.2. Workload Orchestration 59
5.3. Fuzzy Logic Based Workload Orchestrator 62
5.3.1. Two Stage Fuzzy Workload Orchestration 63
5.3.2. Fuzzy Input Variables of the Second Stage 64
5.3.2.1. WAN Bandwidth 65
5.3.2.2. Task Length 65
5.3.2.3. Delay Sensitivity of Task 65
5.3.2.4. VM Utilization 66
5.3.3. Fuzzy Inference System 66
5.3.4. Fuzzy Input Variables of the First Stage 70
5.3.4.1. MAN Delay 71
5.3.4.2. Least Loaded Remote Edge CPU Utilization 71
5.3.4.3. Local Edge CPU Utilization 71
5.3.5. Benchmark Solutions 72
5.4. Performance Evaluation 74
5.4.1. Modeling Networking Delay 75
5.4.2. Simulation Setup 77
5.4.3. Simulation Results and Discussions 80
5.4.3.1. Failed Task Performance 81
5.4.3.2. Service Time Performance 81
5.4.3.3. CPU Utilization Performance 82
5.4.3.4. Task Failure Analysis 84

5.4.3.5. Service Time Analysis 85
5.4.3.6. Application Performance Analysis 86
6. MACHINE LEARNING BASED WORKLOAD ORCHESTRATOR FOR VEHICULAR EDGE COMPUTING 89
6.1. ML-Based Vehicular Workload Orchestrator 92
6.1.1. Multi-Tier VEC Architecture 93
6.1.2. Two-Stage ML-Based Vehicular Edge Orchestrator 94
6.1.3. Classification Models 96
6.1.4. Regression Models 101
6.2. Performance Evaluation 101
6.2.1. Competitor Algorithms 101
6.2.2. Simulation Setup 106
6.2.3. Simulation Results and Discussions 108
7. CONCLUSION 116
REFERENCES 119

LIST OF FIGURES

Figure 2.1. Disadvantages of cloud computing. 8

Figure 2.2. Multi-tier edge computing architecture. 9

Figure 2.3. Multi-access multi-tier VEC architecture. 12

Figure 4.1. Aspects of edge computing simulation modeling. 30

Figure 4.2. EdgeCloudSim block diagram. 34

Figure 4.3. Relationship between EdgeCloudSim modules. 35

Figure 4.4. Generic edge computing architecture. 36

Figure 4.5. Sample XML data for edge server configuration. 39

Figure 4.6. Factory pattern for simulation scenario. 40

Figure 4.7. Complexity analysis of EdgeCloudSim for heavy and light load scenarios. 42

Figure 4.8. Mobility model used in the simulations. 45

Figure 4.9. Measured delay values used in the empirical WAN and WLAN capacity models in the simulator. 46

Figure 4.10. Comparative evaluation of the different edge computing architectures based on all application types. 49

Figure 4.11. Average WLAN Delay based on all application types. 50

Figure 4.12. Average processing time for different application types. 51

Figure 4.13. Performance evaluation of the system for different VM sizes. . . . 52

Figure 4.14. Performance evaluation of the system for different dwell time values. 52

Figure 5.1. Multi-tier edge computing architecture with edge orchestrator. . . 55

Figure 5.2. Proposed scenario representing multiple locations of a campus distributed in a university. 59

Figure 5.3. Workload orchestration flow. 61

Figure 5.4. Two stage fuzzy workload orchestration. 63

Figure 5.5. Fuzzy inference system for the proposed workload orchestrator. . . 64

Figure 5.6. Membership functions for the fuzzy variables representing both network and computational resources. 67

Figure 5.7. Membership function of the consequence classes. 70

Figure 5.8. Membership functions used in the first stage. 72

Figure 5.9. Unified algorithm representing competitor orchestrators. 75

Figure 5.10. Empirical evaluation of WAN/WLAN connections. 77

Figure 5.11. Average failed task based on all application types. 82

Figure 5.12. Average service time based on all application types. 83

Figure 5.13. Average VM utilization based on all application types. 84

Figure 5.14. Distribution of task failure reasons. 85

Figure 5.15. Average service time values for fuzzy based workload orchestrator. 86

Figure 5.16. Average failed task for different application types. 87

Figure 5.17. Average service time for different application types. 88

Figure 6.1. Multi-tier VEC architecture for vehicular networks. 91

Figure 6.2. Multi-tier multi-access VEC architecture and task offloading options. 94

Figure 6.3. Two stage ML-based vehicular edge orchestrator. 95

Figure 6.4. MLP neural network of the *edge* classifier. 97

Figure 6.5. Performance comparison of the classifiers used for the edge scenario. 98

Figure 6.6. SMA based vehicular edge orchestrator. 102

Figure 6.7. MAB theory based vehicular edge orchestrator. 104

Figure 6.8. Game theory-based vehicular edge orchestrator. 105

Figure 6.9. Vehicular mobility model in the simulation. 106

Figure 6.10. Confusion matrix of the classifiers. 109

Figure 6.11. Percentage of failed tasks based on all application types. 110

Figure 6.12. Average failed tasks for different application types. 111

Figure 6.13. Average Quality of Experience based on all application types. . . . 112

Figure 6.14. Histogram of service times based on all application types. 113

Figure 6.15. Service time ratio of the edge and cloud servers. 114

Figure 6.16. Comparative evaluation for different task length. 115

LIST OF TABLES

Table 3.1. Comparison of simulators used in the edge computing systems. . . 21

Table 3.2. Comparison of task offloading approaches for VEC systems. 26

Table 4.1. Applications used in the simulations. 43

Table 4.2. Simulation parameters. 47

Table 5.1. Example fuzzy rules in the knowledge base. 68

Table 5.2. Application types used in the simulations. 78

Table 5.3. Simulation parameters. 79

Table 6.1. MLP Model Parameters. 98

Table 6.2. Comparison of training data sets used by the *edge* classifier. 99

Table 6.3. List of features used for the classification models. 100

Table 6.4. List of features used for the regression models. 101

Table 6.5. Application characteristics. 107

Table 6.6. Simulation parameters. 108

LIST OF SYMBOLS

a_t	Candidate processing unit with minimum utility value
A	Statistical data array
B	WAN bandwidth
B_{min}	Minimum WAN bandwidth threshold
c	Target processing unit (edge or cloud)
c_t	Candidate processing unit for task t
C	Candidate processing unit array
C_c	Capacity of cloud servers
C_e	Capacity of edge servers
$d_{i,c}$	Estimated delay on cloud for the task generated by vehicle i
$d_{i,e}$	Estimated delay on edge for the task generated by vehicle i
$d_{i,max}$	Maximum delay for task generated by vehicle i
$d_{sum}(t, n)$	Service time of candidate processing unit n at time t
$F_\alpha(x)$	Fuzzy value of x based on membership function of α
$F_\beta(x)$	Fuzzy value of x based on membership function of β
$F_\gamma(x)$	Fuzzy value of x based on membership function of γ
$F_\delta(x)$	Fuzzy value of x based on membership function of δ
$I(A_i)$	Index of element i of array A
$k_{t,n}$	Number of tasks that have been offloaded to candidate n up till time t
$L(A)$	Length of array A
O	Target to offload
p_i	Probability of offloading to cloud for task created by vehicle i
P_{ij}	Probability of moving from location i to j
R_i	Delay requirement of task i
S	Offload strategy
SCR_c	Preference score of candidate processing unit c
S_i	Delay sensitivity of task i

$SR_{c,i}$	Average success rate of processing unit c for time period i
T	Incoming task array
T_i	Actual service time of task i
T_n	True negative rate
T_p	True positive rate
$u_{t,n}$	Sum delay of offloading one instruction
U	Average VM Utilization
U_c	Average utilization of cloud servers
U_e	Average utilization of edge servers
U_{max}	Maximum VM Utilization threshold
W_i	Dwell time of user at location i
x_t	Task length
$\tilde{x_t}$	Normalized task length
α	WAN Bandwidth
β	Length of incoming tasks
γ	VM utilization of edge server
δ	Delay sensitivity value of task
λ_i	task arrival rate of vehicle i
μ_α^H	Membership function of α for class *High*
μ_α^L	Membership function of α for class *Low*
μ_α^M	Membership function of α for class *Medium*
μ_β^H	Membership function of β for class *High*
μ_β^L	Membership function of β for class *Low*
μ_β^M	Membership function of β for class *Medium*
μ_γ^H	Membership function of γ for class *Heavy*
μ_γ^L	Membership function of γ for class *Light*
μ_γ^N	Membership function of γ for class *Normal*
μ_δ^H	Membership function of δ for class *High*
μ_δ^L	Membership function of δ for class *Low*
μ_δ^M	Membership function of δ for class *Medium*

μ_c^{cloud}	Membership function of consequence classes for cloud
μ_c^{edge}	Membership function of consequence classes for edge
μ_{cloud}	Fuzzy value for selecting cloud server for all rule
$\mu_{cloud}^{R_i}$	Fuzzy value for selecting cloud server for rule i
μ_{edge}	Fuzzy value for selecting edge server for all rule
$\mu_{edge}^{R_i}$	Fuzzy value for selecting edge server for rule i
ρ	Pricing factor
τ	Constant factor
ω	Output of centroid defuzzifier

LIST OF ACRONYMS/ABBREVIATIONS

3GPP	3rd Generation Partnership Project
AI	Artificial Intelligence
ALTO	Adaptive Learning-Based Task Offloading
AVUCB	Adaptive Volatile UCB
Cloud RAN	Cloud Radio Access Network
CN	Cellular Network
CSM	Constant Speed Movement
CSMA/CA	Carrier-Sense Multiple Access With Collision Avoidance
CSV	Comma-Separated Value
C-V2X	Cellular V2X
DES	Discrete Event Simulation
DRL	Deep Reinforcement Learning
DSRC	Dedicated Short-Range Communications
E2E	End-to-End
EO	Edge Orchestrator
ETSI	European Telecommunications Standards Institute
FLS	Fuzzy Logic System
GI	Giga Instruction
GIPS	Giga Instruction per Second
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
IBN	Intent-Based Networking
ICN	Information-Centric Networking
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IoV	Internet of Vehicles
ITS	Intelligent Transportation System
KB	Kilo Byte

LAN	Local Area Network
LSTM	Long Short-Term Memory
LTE	Long-Term Evolution
M2M	Machine-to-Machine
MAB	Multi-armed Bandit
MAN	Metropolitan Area Network
MBS	Macro Base Station
MCC	Mobile Cloud Computing
MEC	Mobile Edge Computing
MEO	Mobile Edge Orchestrator
MI	Million Instructions
ML	Machine Learning
MLP	Multilayer Perceptron
MMPP	Markov-Modulated Poisson Process
NB	Naive Bayes
OBU	On-Board Unit
OFDMA	Orthogonal Frequency-Division Multiple Access
P2P	Peer-to-Peer
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RSU	Road Side Unit
S3	Amazon Simple Storage Service
SDN	Software Defined Network
SLA	Service Level Agreement
SMA	Simple Moving Average
SMO	Sequential Minimal Optimization
SNR	Signal-to-Noise Ratio
STL	Short-term Load
SUMO	Simulation of Urban Mobility
SVM	Support Vector Machine

TaaS	Transport as a Service
UCB	Upper Confidence Bound
V2I	Vehicle to Infrastructure
V2P	Vehicle to Pedestrian
V2R	Vehicle to RSU
V2V	Vehicle to Vehicle
V2X	Vehicle to Everything
VD	Vehicle Density
VEC	Vehicular Edge Computing
VN	Vehicular Network
VSM	Variable Speed Movement
WAN	Wide Area Network
WLAN	Wireless Local Area Network

1. INTRODUCTION

We are witnessing a tremendous change in the end-user side that generates traffic for today's networks. The proliferation of the commercially available mobile gadgets, sensors, wearable units rendered many novel applications feasible involving IoT, smart home/city, autonomous vehicles, augmented reality, instant video processing, tactile internet, e-health, and Industry 4.0. These applications dictate a totally different modality of workflow when compared with traditional client-server applications served over a cloud computing infrastructure: they create vast amounts of data, require a prompt response (real-time or semi real-time), involve high user mobility and dictate diverse spectrum of requirements. Although this new category of applications is demanding in terms of the expectation of its users, the types of equipment employed for realizing them are getting ever smaller, resulting in stringent battery and computational limits. This, in essence, presents a new challenge where complex software is being implemented on constrained hardware. Edge computing addresses this challenge by boosting the end-user device via computational offloading mechanisms [1] through the use of micro-cloud servers located in the close proximity to the end-user devices.

Cloud computing is one of the vital technologies for accomplishing the aforementioned applications. Since the embedded devices have low computation capacity, they commonly use a remote device to handle the complex operations. For example, smart glasses render several innovative augmented reality applications possible. One of the main bottlenecks of cloud computing for such applications is the network reliability. The Internet connection of such wearable devices may be unreliable due to the unpredictable failures in the wide area network (WAN) communication. Employing a nearby micro cloud server provides better performance compared to a central cloud server, especially for the indoor environments where the clients use a shared Internet connection. These micro cloud servers are used for different purposes on various architectures to solve domain-specific problems such as caching, reducing delay, taking real-time decisions, and bringing the cloud resources closer.

One of the essential criteria of the user expectations from the smart applications is the seamless user experience. For time-sensitive applications such as real-time speech-to-text conversion, face recognition, and simultaneous translation, the users require fast responsivity while communicating with the remote servers. High latency during this communication leads to poor user experience. Edge computing brings several opportunities to overcome this significant bottleneck. It is based on the philosophy that the data should be processed within the locality of its source. Edge computing is entering a new phase where it gains wide acceptance from both the academia and industry as the commercial deployments are starting. Edge computing covers a wide range of the edge related technologies such as, cloudlet based computing [2], Mobile Cloud Computing (MCC) [3], Fog Computing [4], and Multi-Access Edge Computing (MEC) [5]. All these approaches are based on the sound engineering principle that computational resources are better be located on the edge where the real demand is physically present. By doing so, two important technical objectives are achieved: (i) end user devices can get fast access to local computational resources using the local area network (LAN) (ii) the core network is not kept under heavy traffic load, i.e. end user devices' data can be processed or filtered before sending to the cloud.

Edge of the network presents a very dynamic environment with many devices, intermittent traffic, high mobility of the end-user, heterogeneous applications, and their requirements. Generally, the edge computing paradigm provides a clear advantage for time-sensitive applications and mobile environments with no cloud access availability. Executing the whole application or a part of it on the edge of the network brings a significant advantage for the resource-constrained mobile devices, as well. For example, mobile devices with low computational capability can handle complex operations while consuming less battery by utilizing the edge server. In addition, the edge computing provides a lower transmission delay than the cloud computing because the clients do not encounter the WAN delay. As a result, the delay intolerant applications such as real-time face or speech recognition can be executed more efficiently on the edge of the network [6].

Although edge computing is very compelling, it involves a more sophisticated setup technically when compared with the pure cloud computing and mobile computing cases since both computational and network resources are utilized in real-time and in accordance with the stringent requirements. In that respect, it provides a more extensive design space with many parameters, rendering a variety of novel approaches feasible. This creates a considerable challenge in terms of efficiently managing and orchestrating different types of resources based on the dynamic flow of incoming tasks offloaded from the end-user mobile devices or gadgets. Especially, deciding how, when, and where to offload is a complicated process. The rapid change of the status of resources and the need for almost real-time management make it even more difficult.

In this scene, scalable and efficient management and orchestration remains to be a problem. We focus on the *workload orchestration* problem in which the execution locations for the incoming tasks from the mobile devices are decided within an edge computing infrastructure, including the global cloud as well. In this thesis, we name the functional unit, which monitors the rapidly changing conditions and determines the target VM for offloading, as the *workload orchestrator*. The orchestrator decides where to offload the incoming requests by considering many parameters, such as the network latency, data transfer size, and the remote server capabilities. It is an intrinsically hard, online problem [7]. We employ novel orchestration approaches to solve this problem by capturing the intuition of a real-world administrator to get an automated management system. Our approach takes into consideration the properties of the offloaded task as well as the current state of computational and networking resources.

1.1. Thesis Contributions

A growing interest in the edge computing systems yields many approaches that need to be evaluated and optimized. Experimenting on the real cloud environments is not always feasible due to the operational cost and the scalability. Despite increasing research activity during the period that corresponds to the starting phase of this thesis, this field lacked a simulation tool that supports modeling both computational and net-

working resources to handle the edge computing scenarios. Existing network simulators can model network behavior at different levels of granularity. The cloud computing simulators support the modeling and simulation of the computational infrastructures and services efficiently. Starting from the available simulators, a significant programming effort is required to obtain a simulation tool meeting the actual needs. On the other hand, designing a new edge computing tool has many challenges, such as scalability, extensibility, and modeling the mobility, network, and virtualized resources. To decrease the barriers, we designed and implemented a new simulator tool called EdgeCloudSim streamlined for the edge computing scenarios [8]. EdgeCloudSim has exclusively used throughout this study as the primary means for the design, modeling, and simulation of different edge architectures. EdgeCloudSim has actively evolved over the course of this study, where it grew to be a sophisticated, full-fledge discrete-event packet-level simulator that incorporates a wide range of features for the modeling and simulation of edge computing systems. As an open-source project [9], EdgeCloudSim served not only this work but also the community of researchers in this field, and with this respect has been one of the main contributions of this thesis.

Selecting the target server to offload is very crucial and challenging. This decision must be made in real-time by considering many factors. In this thesis, one of the main topics that we focus on is the workload orchestration. Firstly, we propose a fuzzy logic-based workload orchestrator to overcome the inherent orchestration complexity involved in the edge systems [10]. Our proposal is influenced by the Intent-Based Networking (IBN) principles, in the sense that high level "intents" of the administrator expressed in everyday language is incorporated in the decision making process. In order to demonstrate its relative efficiency, our system is comparatively evaluated using the EdgeCloudSim environment. In the experiments, task offloading from a variety of application profiles are considered, both computational and networking resources are modeled to obtain an evaluation as close as possible to the real-world scenarios.

The Intelligent Transportation System (ITS) is another field where edge computing can accelerate the implementation of rich application scenarios. Especially, the

Internet of Vehicles (IoV) vision encompasses a wide range of new intelligent highway scenarios that rely on vehicles with an ever-increasing degree of autonomy and the prospect of sophisticated services like e-Horizon and cognitive driving assistance. Triggered by the self-driving feature, a new generation of passenger profile is being formed where advanced infotainment approaches are expected to leverage the quality of travel. From the technical point of view, in order this vision to become a reality requires a streamlined edge computing infrastructure, namely Vehicular Edge Computing (VEC), where computationally intensive workloads are offloaded to a nearby VEC infrastructure. In order to simulate the VEC environments, we added various new modules such as on-board computing unit, vehicular mobility, and network modules into the EdgeCloudSim.

As a next step in our study, to cope with the VEC requirements, we applied some ML-based orchestration policies for ITS scenarios. As in all ML algorithms, we first trained the models by using the training data set. Generally, obtaining the data set is a tedious process. With the help of our simulator, we can generate training data efficiently and overcome this problem. We used a random orchestration policy under variable load to record the features and labeled output data that can be useful during the training phase of the supervised learning algorithms. Eventually, we trained some classification and regression models to be used in our ML-based workload orchestrator.

ITS consists of highly dynamic environment due to the swift mobility and the wide range of vehicular applications, renders it difficult to efficiently operate a VEC system to yield the crisp performance required on an autonomous vehicle. In this setting, where to offload each task stands out as a crucial problem, and the conventional methods prove insufficient for its solution. Therefore, we proposed a ML-based vehicular workload orchestrator, which takes account of not only the success of the workload but its service time. Detailed performance evaluation via simulation experiments is carried out where both overall and service type-specific performance scores in comparison with opponent schemes are reported.

The contributions of this thesis are listed as follows.

- We introduced an open-source simulator, EdgeCloudSim, for the researchers who are working on the edge computing systems [8, 9].
- We proposed a fuzzy-logic based workload orchestrator for the multi-tier edge computing architectures [10].
- We integrated vehicular edge computing entities and requirements into our simulator.
- We collected training data-set for supervised learning approaches and trained a multilayer perceptron (MLP) neural network for our next study.
- We proposed a machine learning (ML) based vehicular workload orchestrator for multi-tier multi-access vehicular edge computing architectures [11].

1.2. Thesis Outline

The thesis is organized as follows: A background on the edge computing and VEC concepts, as well as the modeling approaches used in these fields, are presented in Chapter 2. In Chapter 3, we present an overview of the related literature.

The main contributions of the thesis are presented in Chapters 4-6. In Chapter 4, we address the requirements of a simulation tool for evaluating the edge computing systems and then introduce our simulator. In Chapter 5, we discuss the orchestrating techniques applied on the multi-tier edge computing architectures and provide our fuzzy-logic based edge orchestrator. In Chapter 6, we state the basic concepts of VEC systems and present our ML-based edge orchestrator for the multi-tier multi-access VEC architectures.

Finally, the thesis is concluded with Chapter 7 by providing a brief discussion of the contributions and possible directions for future research.

2. BACKGROUND

In this chapter, we provide a brief overview of edge computing and VEC. In particular, we describe the essential components of these concepts and how they emerged. We also address the challenges researchers come across in edge computing and VEC. Moreover, we discuss the necessities of the modeling tools to reduce the analysis effort on these systems, and the difficulties encountered in the designs of these tools.

2.1. Edge Computing

The edge computing paradigm has attracted researchers' attention due to creating new opportunities, especially for mobile devices with limited resources. The primary motivation of edge computing is moving the complex operations to the nearby micro-cloud servers located at the edge of the network. Edge computing makes it possible to improve the quality of service (QoS) regarding the responsiveness and battery lifetime of the mobile devices by using the resources in the proximity, which can be a network resource or a computational resource in between the client and the cloud datacenters. Some examples of the complex operations which cannot be adequately handled on the battery-powered devices with little computing power are; real-time video processing, voice to text conversion, and big data processing. Cloud computing is one of the conventional solutions to provide such applications on these powerless devices. However, it has some limitations. Firstly, the broadband internet connection required for cloud access generates a higher network delay compared to the local communication. Secondly, WAN generally provides a less reliable connection than LAN. The network must be both reliable and fast for scenarios such as real-time object detection in smart manufacturing. Edge computing has been proposed to overcome these challenges. The drawbacks of the conventional solutions are illustrated in Figure 2.1.

Edge computing, cloudlet based computing, fog computing, MCC, and MEC are, in essence, similar concepts. In this thesis, we focus on the term edge computing to

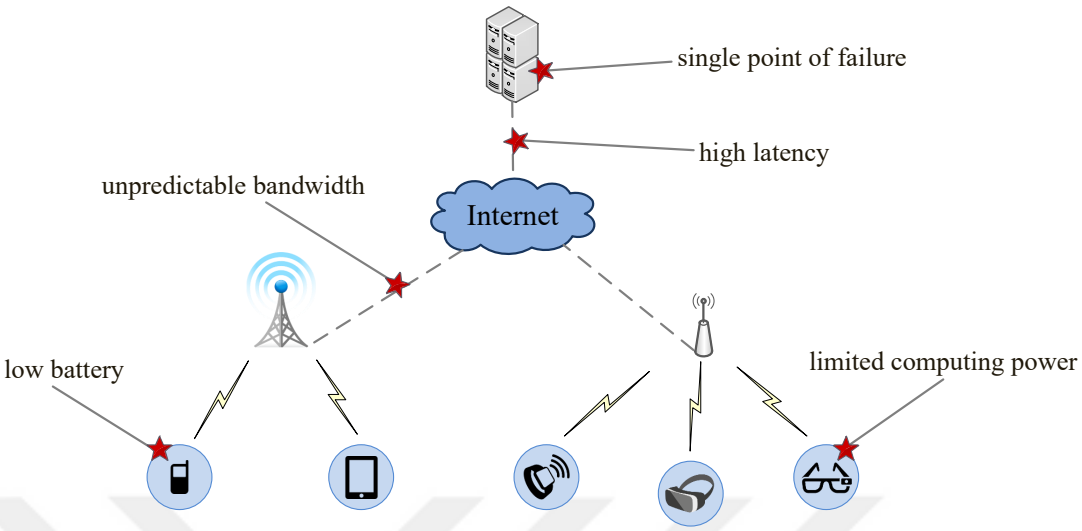


Figure 2.1. Disadvantages of cloud computing.

maintain consistency. The history of edge computing paradigms is explained in [1]. The Cloudlet concept formed in the light of Satyanarayanan's definition was the first study in this domain [2]. The cloudlets can be considered as the micro-clouds in the proximity, and the mobile users deploy and manage their own VMs over wireless networks. Among the various edge computing approaches, the Cloudlets deserve further attention. Then Cisco proposed Fog Computing [4], which has a similar idea with Cloudlets. Fog computing architectures generally consist of many devices that are located somewhere on the edge of the network and capable of wireless communication. Finally, these approaches attracted the attention of the telecommunication service providers. European Telecommunications Standards Institute (ETSI) proposed the Mobile Edge Computing (MEC) concept to speed up the expansion of edge computing in the cellular networks [12]. Their main motivation is providing real-time access to the services by bringing the edge and cloud computing capabilities into the edge of the radio access network (RAN). In order to cover not only the cellular networks but also different access technologies such as WiFi, ETSI changed the *mobile* term in the MEC to *multi-access* by keeping the abbreviation the same. The access points could be WiFi, cellular, or a mix of both access technologies in Multi-Access Edge Computing architectures. A typical edge computing architecture and end-user devices are depicted in Figure 2.2.

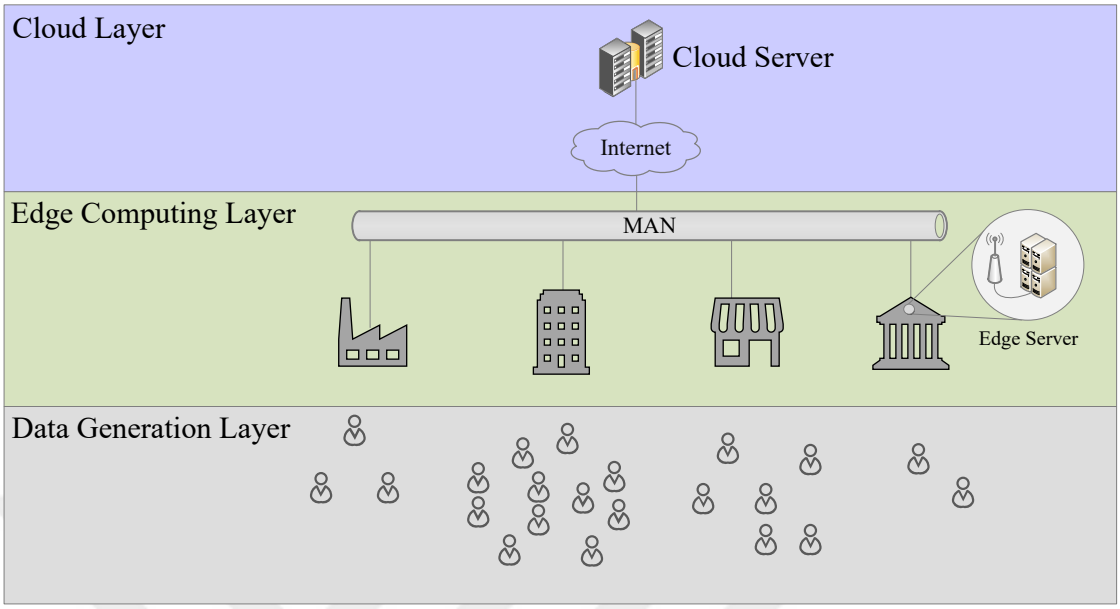


Figure 2.2. Multi-tier edge computing architecture.

Internet of Things (IoT) is another emerging technology for edge computing [13]. Today the number of small embedded devices connecting to the Internet is increasing. These devices facilitate the development of IoT applications such as home automation, healthcare, transportation, and environmental monitoring. Since the embedded IoT devices have low computation capacity, they commonly use a remote device to handle compelling operations. Not only IoT devices but also the wearable devices benefit from cloud computing [14,15]. The main bottleneck of cloud computing for IoT applications is latency. For example, a face recognition or real-time speech processing application running on the smart glasses [16] requires a high data rate. However, the Internet connection speed of the smart glasses or similar mobile wearable device may change due to the unpredictable delays in WAN communication. For such kinds of applications, the edge computing-based solutions could provide a satisfying user experience. As a result, IoT can bring plenty of potential use-cases for the edge computing systems.

2.1.1. Challenges of Edge Computing

The edge computing concept has some challenges besides the proposed opportunities. We explored following open research issues for edge computing as a result of our extensive literature survey.

- A mechanism for task offloading to a remote server, which is called as *how to offload* problem, is one of the significant challenges. There are different approaches to solve this problem. In some studies, a virtual machine image of the client (mobile) device is generated and then sent to the server. Some authors propose application partitioning mechanisms to offload related threads or processes. However, this brings further problems because partitioning an application is not a straightforward task. An application can be divided either statically on compile time (mostly by using annotations) or dynamically on run time. Dynamic application partitioning is a more compelling option.
- In addition to the task offloading methodology, the granularity of task offloading is also a crucial problem. The offloading frequency is directly proportional to the granularity of offloading. Therefore, we often call this problem as *when to offload* problem. In order to increase the overall system performance, many metrics, such as the task execution time, energy, and cost, should be taken into consideration while dealing with this problem.
- The resource provisioning is another important open research issues for the edge computing systems. This problem is often described as *where to offload* problem. Deciding the target VM requires many constraints, such as the service level agreement (SLA) requirement, energy consumption, and the average utilization of the servers. Where to offload problem is an online problem and does not have a unique solution. Therefore different heuristics and more complex approximation techniques are used. In this thesis study, we work on this problem and propose fuzzy logic-based and ML-based strategies.
- Supporting *handoff* between the edge servers is another issue arising from the mobility of the clients. Unlike the clients utilizing conventional cloud computing

resources, edge computing users are mobile and battery-powered. To increase the overall performance, the clients should change their serving edge server regarding several metrics such as the signal strength, bandwidth, delay, and energy consumption. The problem is more complicated for the IoT scenarios because the end-user devices use different protocols like WiFi, Bluetooth, and Zigbee. As a result, providing a seamless service and handoff among the edge servers is a challenging problem.

- Synchronizing client's data among multiple servers is also a challenge for the edge computing systems. In some cases, the clients' data on the VM or the whole VM should be migrated to another server. The data synchronization problem, or the more commonly mentioned *VM migration problem*, is mostly caused by the mobility of the clients. If the mobile device moves from one edge server's coverage to another, its resources should also be moved to ensure a seamless transition. This requirement may also arise to utilize server resources more effectively.
- Last but not least, interoperability and portability issues should be studied by the researchers. From the edge server point of view, executing the same code on different CPU architectures is a difficult job. From the end-user equipment point of view, supporting different operating systems is also an obstacle. In addition, heterogeneity in terms of the mobile device type, their communication interface, and their battery consumption complicates the problem further.

2.2. Vehicular Edge Computing

The VEC-based systems provides many opportunities for the next generation vehicular applications to provide an improved driving experience. In this regard, executing the computationally intensive applications which have strict delay requirements on the vehicles is becoming a crucial problem. The edge computing philosophy is a promising paradigm to provide enhancements for the vehicular services by bringing the computational resources closer to the vehicles. The main idea behind using the resources at the edge of the network is to fulfill the needs of the complex applications with a strict delay requirement such as driving safety assistance and natural language

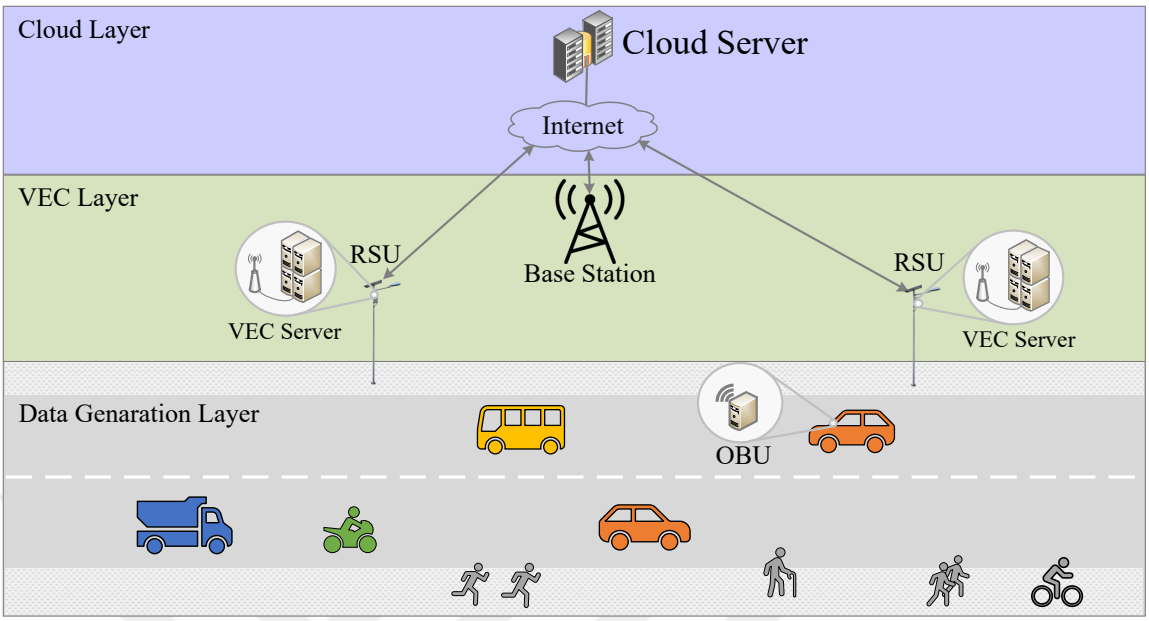


Figure 2.3. Multi-access multi-tier VEC architecture.

processing. Compared to the conventional cloud computing-based solutions, the VEC systems can provide more context and location-aware services, better mobility support and lower deployment cost and latency. On the other hand, cloud computing services can provide highly scalable storage and computing capability.

The VEC architecture generally includes the vehicles, edge servers, and the cloud servers as shown in Figure 2.3. It can be considered that there are three tiers available in this architecture. The vehicles are located at the first tier and they have computational resources that are provided by their On-Board Unit (OBU). If required, some of the operations can be handled by OBU locally. The second layer consists of the components called Road Side Unit (RSU). The edge servers are deployed on RSUs. The third layer includes the traditional cloud computing components. The cloud datacenters and related servers are operated in this layer.

In general, all the interactions between the vehicular entities are called Vehicle to Everything (V2X). The vehicles can perform the task offloading process via V2X

communication technologies. V2X is a broad term that refers to the different communication models used by the vehicles, such as Vehicle to Vehicle (V2V), Vehicle to Pedestrian (V2P), Vehicle to RSU (V2R) and Vehicle to Infrastructure (V2I) [17]. V2V and V2P provide a peer-to-peer (P2P) interaction among the vehicles and pedestrians. On the other hand, V2R and V2I make it possible for the vehicles to communicate with the infrastructure which can be considered as the service provider. Different organizations proposing different solutions to standardize V2X communication models. 3rd Generation Partnership Project (3GPP) pushes the Long-Term Evolution (LTE) and 5G standards to enable the V2X services [18]. On the other hand, the Institute of Electrical and Electronics Engineers (IEEE) and ETSI propose the IEEE 802.11p standard for accessing the V2X services. Both standards have their own advantages and disadvantages [19]. The cellular network is not convenient for the V2V and V2P services since it has some limitations for the P2P transmission. In this regard, 802.11p provides the Dedicated Short-Range Communications (DSRC) standard which is a better solution for the P2P scenarios and so for the V2V and V2P services. When it comes to the V2I and V2R services, the cellular networks provide better services due to their reliability and coverage capabilities while 802.11p may not provide sufficient service when the vehicles are moving in a fast manner. As a result, when considering the requirements for all V2R, V2V, V2P, and V2I scenarios, it can be concluded that a hybrid network infrastructure including LTE/5G and 802.11p could be the optimum architecture for VNs [20].

2.2.1. Challenges of Vehicular Edge Computing

Beyond the aforementioned opportunities, VEC based solutions have some challenges which are listed below:

- The first challenge is the deployment of edge servers and the capacity planning [21]. Since the number of vehicles in the road is variable, it is important to decide how much resources should be deployed on which location.
- The second challenge is the development of a methodology for task offloading [22].

The vehicles can offload their tasks to the other vehicles, edge or cloud servers. Hence, there should be a framework that makes the offloading process easier.

- The third challenge is the determination of the offloading strategy [23]. A vehicle can process a task locally or send it to another computing device depending on the available resources. The answer of when and where to offload may have a considerable impact on the overall system performance.
- The fourth challenge is the efficient wireless resource allocation [24]. The vehicles can use different V2X access technologies such as cellular V2X (C-V2X) and the DSRC, which is based on the IEEE 802.11p. Vehicles can also interact with other vehicles through V2V links. The heteroginyty of wireless access technologies and different QoS requirements of the services make resource allocation a challenging process. The high mobility of the vehicles in different directions increases this challenge even more.
- The fifth challenge is applying the end-to-end (E2E) slicing techniques on the VEC systems [25]. The network-slicing concept is proposed to overcome the extensibility problem of one-size-fits-all networks. MEC and Software Defined Network (SDN) technologies are the potential paradigms used in the network slicing. The E2E slicing concept also brings the necessity of slicing not only the network resources but also the computational resources [26].
- Last but not the least challenge is security and privacy issues. Most of the use-cases require data sharing among the vehicles so that the security and privacy problems should be addressed and resolved [27].

2.3. Modeling the Edge Computing Systems

Today, edge computing is a rapidly emerging research topic where many researchers propose new architectures and methods to improve current models. The increasing trend towards the edge computing paradigms has led the researchers to study new internet architectures and propose new approaches. The performance of the system should be analyzed by taking many aspects into consideration to evaluate and optimize the proposed architectures. However, the design and development of an edge

or fog computing architecture may not always be reasonable due to the deployment difficulty of the datacenters and clients which are mostly mobile. In addition, the cost of experimenting with real cloud environments is another factor pushing the researchers to search for other performance evaluation approaches. As a result of these difficulties, the researchers generally tend to use the experimental testbeds or the simulators to evaluate and optimize the proposed computing architectures.

Edge computing based solutions can be obtained in multiple domains using different techniques. Given that there are no de facto standards and that there is an abundance of edge architectures emerging in the literature, tools for modeling and analyzing the edge systems are required. One option to model the edge systems is to implement testbeds that are specific to the requirements of a use case. Given the availability of open-source tools to virtualize resources (computational and network), it would be feasible to develop testbeds for a research environment. For example, the Living Edge Lab [28] is an experimental testbed. However, setting up and maintaining testbeds can be quite expensive. Additionally, for a complete performance analysis, testbeds, and sometimes even real-world deployments may not lend themselves to repeatable and scalable experiments as could be obtained using simulators [29]. Therefore, simulators are employed to complement experimental testbeds for a thorough evaluation.

2.3.1. Challenges of Modeling Edge Computing Systems

At the heart of a simulator is a complex mathematical model that captures the environment. Although simulators are advantageous, numerous modeling challenges will need to be addressed in designing an ideal (or even a reasonable) simulator [30]. An ideal simulation environment should incorporate programming APIs, management of configuration files, and UI dashboards for easy modeling with minimum manual effort. We anticipate that the same principles would apply to an Edge simulator. In this section, we consider five specific modeling challenges that will need to be addressed for an ideal Edge simulator.

- The first challenge is related to computational resource modeling. Like a Cloud datacenter, an Edge server will provide computational power to its users via virtualization techniques. A simulation environment in this context should support the creation, resizing, migration of virtual resources, and model CPU, memory, and network resource consumption at different levels of granularity (process, application, and entire node). The model will need to capture the possibilities of using existing traffic routing nodes, dedicated nodes, or a combination of these.
- The second challenge is related to the modeling of demand. To be able model the load on an Edge computing system, the demand on the Edge resource due to an individual user (or a collection of users) will need to be modeled. Accounting for the heterogeneity of mobile devices and the traffic generated by a variety of applications is complicated. End-user devices or a Cloud server may offload compute on to Edge servers, and this will need to be accounted for in a demand model. The distribution of demand and the inter-arrival times of traffic on the Edge will need to be considered. Profiles of the users and family of applications with predefined distributions would also be beneficial.
- The third challenge is related to mobility modeling. Mobility is a crucial component that will need to be considered for accurately modeling the time-varying demands on the Edge. The need for mobility arises in multiple use-cases. For example, a human with a wearable gadget moving from the coverage area of one Edge server to another could result in the migration of the service from one edge server onto another or the replication of the service with the user data on another edge server. In such a use case, the simulation environment should allow for designing experiments that realistically capture mobility in a variety of forms.
- The fourth challenge is related to network modeling. Performance and behavior patterns of the network are critical for the overall operation of an Edge system. Accurate network delay modeling will not be easy due to dynamic workloads that operate using different network access technologies, such as Wi-Fi, Bluetooth, and cellular networks. In contrast to legacy network simulators, an Edge simulation tool should be able to scale network resources rapidly. This requirement arises due to the slicing approach described previously in which multiple network slices

will need to be modeled in the network [31].

- The final challenge considered is simulator efficiency. Simulators will need to be scalable, extensible to changing infrastructure requirements, and easy to use. Taking into account the upcoming IoT and Machine-to-Machine (M2M) communications, the time complexity of simulators accounting for federated Edge resources should model the connections of a large number of devices and users.

3. LITERATURE REVIEW

The edge computing simulator and the workload orchestration problem on edge computing systems are the main contributions of this thesis. In this chapter, we give an overview of the existing works on these topics and present their strengths and weaknesses.

3.1. Related Work on Edge Computing Simulators

Many cloud computing simulators have been proposed as a result of the growing popularity of cloud computing [32]. In essence, these simulators provide the computational models for the virtualized cloud environments, and they are beneficial for the CPU, memory, and storage models, as well as the energy consumption models. Some popular cloud simulators are CloudSim [33], GreenCloud [34], iCanCloud [35], CloudSched [36], and SimGrid [37].

CloudSim is mainly designed for modeling Infrastructure as a Service (IaaS) cloud computing environments [33]. It supports both modelings of the cloud components such as datacenters, hosts, and VMs and the resource provisioning policies such as CPU, storage, memory, and bandwidth utilization models. CloudSim is the most commonly used cloud simulator for typical cloud computing scenarios [38]. In conventional cloud computing, the clients request from the datacenter to create a list of VMs and run a list of tasks. When the datacenter receives such a request, it tries to create the VMs and executes the related tasks. The tasks are considered as a bunch of operations that keep the VMs busy for hours. This flow of operation is similar to renting a VM from the datacenter operators such as Amazon [39] and Microsoft [40] for a certain time. However, the edge computing scenarios differ from the cloud computing scenarios in the sense that the tasks do not require too much time to be executed. The tasks offloaded by the mobile devices to the edge server is generally processed in sub-second intervals. Due to the small size of the tasks, creating a new VM for each request is not efficient.

GreenCloud [34] is an extension of NS2 [41] simulator. The primary motivation of GreenCloud simulator is to capture the energy usage of the computing and communication elements. Being an extension of NS2 provides GreenCloud to implement the full TCP/IP protocol reference model. Implementing a full TCP/IP protocol reference model enables detailed modeling of the energy consumption on the network switches and links. However, this brings an overhead to the system regarding memory usage and the simulation time, which creates a disadvantage for GreenCloud.

iCanCloud [35] is an OMNeT++ [42] based simulator platform. The main aim of iCanCloud is to model and simulate the large environments with thousands of nodes. The performance and scalability are the primary objectives of iCanCloud, so that it supports executions of parallel simulations. Like the other simulators, it allows modeling of the computational aspects of the virtualized cloud environments. iCanCloud provides a graphical user interface (GUI) to describe the simulation scenario.

CloudSched [36] is a lightweight cloud simulator that is implemented on the Java platform. The most considerable advantages of CloudSched are less computing time and memory consumption when compared to the other simulators. It can simulate the large scale IaaS cloud computing environments due to its lightweight design and scalability. CloudSched can model the datacenters, VM requests, and VM scheduling policies. The main focus of CloudSched is VM scheduling policies and algorithms in the cloud datacenter at IaaS layer instead of task scheduling in the VMs. Like iCanCloud, CloudSched also provides a GUI to customize the simulation configuration.

SimGrid [37] is used for distributed applications that are executed on distributed platforms. It is initially designed for grid computing-based applications, but it can be used for simulating a variety of computing use cases such as volunteer computing, cloud computing, fog computing, P2P systems.

The edge computing systems have different characteristics in terms of the users and the network topology compared to the cloud computing systems. The users of

the edge computing architectures are mobile, but cloud computing simulators do not consider mobility. The E2E transmission delay is generally considered as a static value in cloud computing simulators. However, the communication between mobile users and edge devices requires wireless interfaces, which create a necessity for more detailed network modeling. Besides, the applications utilized in edge computing also have different characteristics when considering IoT devices. In order to cover all the mentioned different requirements, new simulators focusing on the fog, edge, or IoT application scenarios are required. IOTSim [43], SimIoT, [44] and iFogSim [45] are some of the examples of these simulators.

IOTSim [43] is another simulator that extends CloudSim. It is proposed for simulating the edge computing environments where a massive amount of data is sent to a big data processing system by the IoT application. It adds the storage and the big data processing layers into the CloudSim in order to support a big data processing system. In the storage layer, the network and storage delays are simulated for IoT applications. When it comes to the big data processing layer, it simulates MapReduce model to support the batch-oriented data processing paradigm.

SimIoT [44] is developed by extending the SimIC [46] simulator. It adds an IoT layer to the SimIC in a way to allow the desired number of IoT devices to request for cloud resources. SimIoT has two critical modules, namely the IoT device input system and the communication broker. The IoT device input system is responsible for gathering the sensor events and redirecting them to the communication broker. The communication broker converts the sensor data to the contextualized information and generates a request to be sent to the default SimIC cloud entities.

iFogSim [45] runs on top of CloudSim. It is developed for simulating IoT and Fog environments by modeling components like the sensor, actuator, fog devices, and the cloud. iFogSim allows the definition of the location of devices getting service from the fog servers. However, this information is static, and the mobility model does not update it. The network model in iFogSim is over-simplistic, where a fixed delay is

Table 3.1. Comparison of simulators used in the edge computing systems.

Ref	**Underlying Simulator**	**Language**	**Year**	**Strengths**	**Weaknesses**
[33]	None	Java	2010	easy to model cloud computing infrastructures and services	lack of modeling mobile clients and their communication model
[34]	NS2	C++	2016	can model a wide variety of communication protocols	implements a simplistic application model
[35]	Omnet++	C/C++	2011	can simulate the large environments with thousands of nodes	designed for cloud computing scenarios, edge or fog scenarios are not considered
[36]	None	Java	2015	less computing time and memory consumption	mainly focuses on the resource scheduling algorithms on cloud datacenters
[37]	none	C/C++	2001	useful for scheduling algorithms for distributed applications	edge computing use cases are not considered
[43]	CloudSim	Java	2016	can simulate MapReduce big data processing model efficiently	focuses only on big data processing perspective, edge computing is not addressed
[44]	SimIC	Java	2014	suitable for IoT application scenarios where many sensors send their data to the cloud	only focuses on cloud infrastructures, edge computing scenarios cannot be implemented
[45]	CloudSim	Java	2016	efficiently simulate sensors, actuators, fog devices, and cloud	simplistic network model which cannot simulate mobile clients realistically

assigned to links ignoring the effect of network load on the operation. As discussed above, for the realistic assessment of edge computing proposals, network resources, as well as computational resources, should be considered with satisfactory accuracy.

To the best of our knowledge, an edge computing simulator covering the concerns mentioned above has not yet been proposed. Therefore, we propose EdgeCloudSim open to the research community as a sophisticated tool based on CloudSim for the performance evaluation of the edge computing proposals. The strengths and weaknesses of the other simulators are briefly compared in Table 3.1.

3.2. Related Work on Workload Orchestration for Edge Computing

Orchestration in edge computing is a broad concept that is comprised of several management efforts on multiple levels. The mobile edge orchestrator (MEO) proposed in the ETSI MEC reference architecture has comprehensive responsibilities such as managing the available computing/storage/network resources and the life cycle of the applications [47]. However, an edge orchestrator can also have a confined scope instead of serving the entire system. As an example, Hegyi et al. propose the concept of application orchestration, which solves the application deployment problem on the edge [48]. Karagiannis et al. propose an edge computing orchestrator that decides which application should be placed on which edge server [49]. Santoro et al. propose the workload orchestration concept that solves the workload placement issue in the fog computing environments [50].

Baktir et al. propose a service-centric approach at the edge servers by using the orchestration capabilities of SDN [51]. As a solution for tracking the services offered by the edge server in highly dynamic and mobile conditions, a service-centric edge computing solution based on SDN is used. SDN decouples the data plane from the control plane and provides networking capabilities via the north-bound interface. Two of the significant components of their proposed system, the orchestration and load-balancing, are implemented by the north-bound applications. In their system, the

orchestrator takes the list of IP addresses and forwards it to the load balancer. The load balancer north-bound application decides on the destination server and sends that server's IP address to the controller.

Another approach presenting an effective cloud orchestration is FocusStack [52]. In their study, the authors address the problem of allocating and managing resources with an approach to manage the edge devices as IaaS clouds. They offer intelligent geo and context-aware messaging bus to define the devices which have to be focused by checking the device location, edge device health, and the capabilities as well as the user authorization preferences. By this method, they minimize the resource utilization of the edge devices and make the cloud control plane to be more efficient and scalable. As a result, the constantly moving devices which are disconnected from the network are simply excluded from the focus of attention of the current orchestration task.

Pham et al. offer a cloud-fog computing system where the fog nodes are owned by a fog provider, and the cloud nodes are rented from the cloud providers [53]. The authors' main aim is to execute the users' large-scale offloading applications efficiently. They propose a heuristic-based algorithm for the task scheduling problem. Their primary focus is the monetary cost and the makespan for the execution of the tasks.

Simoens et al. identify the challenges of orchestration in distributed edge clouds [54]. They focus on the composite services where the components are running in the different service nodes. In their work, a two-layer network for service orchestration and instance selection based on FUSION architecture are studied. The authors propose the 'service resolution layer' as an overlay of IP-layer, which resolves the service requests to the locator of the best instance among many replicas of the same service components running throughout zones.

Bittencourt et al. discuss the workload offloading on the edge/fog computing infrastructures [55]. The application requests are served either in the edge or cloud according to three different policies. The decision can be based on the CPU capacity

and delay requirements, but they use a static network delay for the WLAN, MAN, and WAN communication.

Maio et al. [56] propose and evaluate a task offloading scheme that focuses on how an application composed of sub-tasks can be deployed over an MCC infrastructure with simplistic networking assumptions. Our work, on the other hand, focuses on a variety of applications being simultaneously run from numerous mobile devices and gadgets on the whole system within a realistic setting that connects edge servers through MAN whose performance varies as a function of the current load.

The benefits of utilizing cloud computing and its central role on the offloading decision are explained in [57]. The authors state that, although the code offloading approach is studied largely on the mobile cloud context, many real cloud computing advantages are still not exploited properly. The overall offloading decision can take advantage of cloud computing's characteristic features, such as performance metrics, parallelization of tasks, and elasticity. To accomplish this, they proposed a fuzzy decision engine for offloading, which considers both the mobile and the cloud variables.

Another study proposing a fuzzy logic-based code offloading method is done by Hosseini et al. [58]. In their work, the offloading ratio is calculated via a fuzzy logic-based method by using parameters link delay and Signal-to-Noise Ratio (SNR). The authors demonstrate that using the fuzzy logic approach provides better results than using a weighted average of the two offloading parameters used. Similar to the previous fuzzy logic-based approach, this study also is not a direct candidate to comply as a rival algorithm since it operates at different layers, focusing on different decisions.

3.3. Related Work on Workload Orchestration for Vehicular Edge Computing

The task offloading is one of the most important features used in the VEC systems. The workload orchestration particularly solves the where to offload problem,

and it has already been studied for the edge computing systems. Cao et al. discussed different task offloading approaches for the MEC architectures [59]. They showed that using the traditional methods may be useful for a static or slowly varying environment, but decision making is difficult on the fast-changing mobile networks. They recommended using ML-based solutions for such heterogeneous environments where the state of resources changes rapidly and cannot be predicted precisely. As a result, Cao et al.'s work inspired us to develop an ML-based workload orchestrator in the VEC domain. The other studies focusing on where to offload problem in the VEC systems are explained in this section and briefly compared in Table 3.2.

There are very few studies in the literature using ML-based task offloading approaches for the VEC systems. Ning et al. [60] proposed a deep reinforcement learning (DRL) based solution. Reinforcement algorithms have four key elements, which consist of agents, system states, actions, and rewards. When the agent performs an action, the system state changes, and a reward is received depending on the result of the action. The most challenging part of operating DRL-based solutions on the VEC systems is determining the actions and states because the VEC environments are highly dynamic, and the network and computing resources fluctuate rapidly. Ning et al. overcome this problem by using a reply buffer that stores previously discovered state, action, reward, and next state values in a large table. As a result, the reward and the next state are immediately observed whenever the target RSU is chosen. However, preparing the reply buffer is a complex operation. We also tried reinforcement algorithm-based workload orchestrator in our study, but it provides poor performance compared to the other supervised-learning methods, which are using tagged historical data.

Sun et al. introduced an adaptive learning-based task offloading (ALTO) algorithm for the dynamic and uncertain VEC systems [61]. As they stated, the fast varying wireless channel and computing resources bring extra challenges to the offloading problem. They propose a multi-armed bandit (MAB) theory-based solution. MAB theory is commonly used to learn unknown environments. ALTO works in a distributed manner and tries to minimize the average offloading delay. Although their algorithm only

Table 3.2. Comparison of task offloading approaches for VEC systems.

Ref	**Methodology**	**Mobility Model** [1]	**# of Vehicles**	**Hand-over**	**Cellular Access**	**Network Congest.**	**V2V Offload**	**V2R Offload**	**V2I Offload**	**Simulation Tool**
[60]	DR Learning-based	DRJ	10	✗	✓	✗ [2]	✗	✓	✗	Python
[61]	Multi-armed bandit based	SUMO	9-18	✗	✗	✗ [3]	✓	✗	✗	Matlab/Veins
[62]	Game theory-based	N/A	6-70	✗	✓	✗ [3]	✗	✓	✗	Matlab
[63]	Matching theory based	SUMO	4-8	✓	✓	✗ [5]	✗	✓	✓	Matlab
[64]	Distributed greedy alg.	CSM	No data	✗	✗	✗ [6]	✓	✓	✗	No data
[65]	Genetic algorithm based	No data	20-120	✗	✗	✗ [7]	✓	✓	✗	CloudSim
[66]	Dist. approximation alg.	CSM	40	✗	✗	✗ [8]	✗	✓	✗	No data
Ours [9]	Machine learning based	VSM	100-1800	✓	✓	✓	✗	✓	✓	EdgeCloudSim

[1] The mobility of vehicles can be simulated via traffic simulation tools like SUMO, or implemented via a custom movement model; DRJ = Discrete Random Jumps, VSM = Variable Speed Movement, CSM = Constant Speed Movement, N/A = Static Vehicle Position.

[2] OFDMA technology is utilized for the links between vehicles and RSUs, and there is no interference between vehicles on this link.

[3] The wireless channel state remains static during the uploading process.

[4] The link model uses a frequency-flat block fading Rayleigh channel, where the block length is more than the maximum service time.

[5] Each vehicle allocates an orthogonal channel (no interference among the vehicles). In addition, the WAN delay is a constant value.

[6] Each vehicle uses an independent and identically distributed channel. Only the distance is considered.

[7] Each vehicle transmits data with fixed power and constant data transmission rate.

[8] Only one task can be transmitted in a time slot without collision via CSMA/CA based data transmission.

[9] This work is one of the contribution of this thesis and will explained in detail in Section 6.

considers the V2V offloading, we adapted Sun et al.'s MAB theory-based solution to our multi-access multi-tier VEC architecture as a qualified competitor algorithm. The differences and modifications made for the adaptation are explained in Section 6.2.1.

Wang et al. utilize a game theory-based approach to find the offloading probability of each vehicle in vehicular MEC networks [62]. Their non-cooperative computation offloading game model operates a best-response algorithm that tries to maximize the utility of each vehicle. The vehicles adjust their offloading probability by considering the offloading probability of other vehicles in the previous stage. They showed that this strategy could converge to the unique Nash equilibrium. Their game theory-based solution uses a lightweight and distributed algorithm. We implemented this algorithm on our simulation tool and employed it as a competitor algorithm by applying slight changes as explained in Section 6.2.1.

Liu et al. used a matching-based task offloading approach to minimize the network delay [63]. It is claimed that the matching theory can provide the decentralized and efficient solutions for the complex and dynamic network topologies. In their work, the VEC architecture includes three layers, which include the vehicles, RSUs, and a macro base station (MBS) respectively. The MBS is one of the most critical entities of their system. All the vehicles and RSUs are connected to the MBS, which is responsible for the task offloading and handover operations. The matching algorithm works iteratively by sending a matching request to the RSUs by the vehicles. We think that these requests create an additional overhead on the system and slow down the offloading decision. They performed a simulation study on Matlab by using the data obtained from the Simulation of Urban Mobility [67] (SUMO) tool. Therefore, it can be said that the evaluation was carried out on the realistic road topologies. However, the network model is not realistic since the WAN delay is assumed as a constant value.

Wang et al. proposed a federated offloading scheme and a distributed algorithm to minimize the service time on the VEC systems [64]. The federated offloading scheme divides the tasks into three parts; one part for the local computation, one part for the

edge server through the V2I communication, one part for the idle vehicles through the V2V communication. The allocation ratio among these three parts and the offloading order considering the edge servers and neighbor vehicles affect the total latency. The authors formulate the communicational and computational models used in their scenario and solve this latency-minimization problem. The distributed algorithm is operated to obtain an optimal route among the neighboring vehicles. This algorithm uses a greedy method to find a vehicle that promises the lowest service time. We think the most challenging part of the federated offloading scheme is to divide the tasks at different rates. This rate can be adjusted in the optimization formula easily, but it will be challenging to apply in real life.

Xu et al. state that smart vehicle applications require computation-intensive tasks [65]. They propose a computation offloading method named V2X-COM which defines the task offloading process as a multi-objective optimization problem and uses a genetic algorithm to decrease the latency while increasing the resource utilization. V2X-COM offloads the task either to the edge servers or the other vehicles. Since the demand of vehicles is not known in advance, the task offloading is an online problem which requires a low-complexity solution. The complexity of the genetic algorithms is known to be high; hence it may not be sufficiently fast for this problem.

Dai et al. address that deciding the target VEC server and determining the offloading ratio is an NP-hard problem [66]. They formulated the VEC server selection and load balancing problem as a mixed-integer nonlinear programming problem, then propose a low-complexity approximation algorithm to solve this problem. The algorithm runs on the vehicles in a distributed manner. In their model, some parts of the tasks can be executed locally on the vehicle, and the rest can be offloaded to the VEC server. Dividing the tasks and combining the results may not be favorable for some applications which require the response in a real-time manner. For example, an image processing based accident prevention application needs to receive a danger signal as quickly as possible. The solution mentioned above may not be very suitable for these kinds of applications.

4. EDGECLOUDSIM

There is an increasing trend towards edge computing, and many researchers are devising new schemes for the implementation of new approaches. A crucial challenge in proposing a new edge computing solution is to realistically assess the performance of the envisioned system while considering the relevant engineering parameters. The researchers can use the real cloud environments [68], experimental testbeds [69] or simulators [70] to evaluate their approaches. There are pros and cons to the selection of these options. Using a real cloud environment is not a cost-effective option due to the deployment requirements. On the other hand, working on the experimental testbeds brings in difficulties regarding the repeatability of the experiments and scalability of the proposed architectures [71]. In addition, the clients of the edge computing systems are mostly mobile. Operating mobile clients on real cloud environments or testbeds is a challenging issue. Therefore, the researchers generally prefer to use simulators to evaluate and optimize their edge computing approaches and designs.

A unique requirement specific to the edge computing systems is the need to consider the mobility, network, and computational resources when evaluating their performances. To the best of our knowledge, no simulation environment within the current solutions handles these concepts efficiently and easily. Figure 4.1 depicts the main aspects of the edge computing simulation modeling in independent axes. Conventional network simulators such as NS2 [41], Omnet++ [42] and Riverbed Modeler [72] can model various types of network protocols and topologies effectively. However, the cloud computing elements and scenarios are not handled in these tools. Therefore, a significant implementation effort is required for modeling the computational aspects relevant to edge computing. When it comes to cloud computing simulators, they allow the modeling of cloud computing infrastructures and application services. One of the most popular simulation tools streamlined for the cloud computing setup is CloudSim [33]. The main drawbacks of CloudSim from the edge computing point of view are (i) lack of the dynamic WLAN and WAN communication models, (ii) lack of

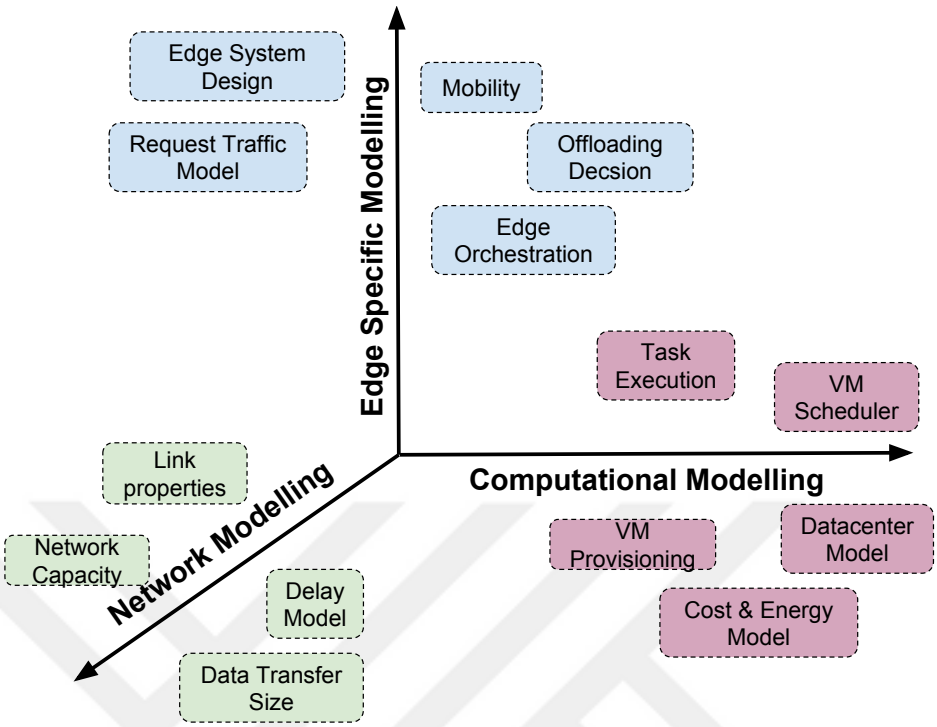


Figure 4.1. Aspects of edge computing simulation modeling.

the mobile nodes and mobility support in general, (iii) lack of the realistic edge type load generator model. Generally, the researchers extend CloudSim by implementing the missing parts concerning the needs of their subject matter. For example, IOT-Sim [43] and iFogSim [45] are implemented by extending CloudSim. Although they are useful for generating relevant workload of IoT and fog computing scenarios, the mobility and dynamic network model are not considered by these tools. When all the shortcomings mentioned above are taken into consideration, the necessity of a new tool that can satisfy the need for realistic simulation of the edge computing scenarios can be seen.

In this chapter, we introduce a new simulator called EdgeCloudSim [73], which covers the whole capability space presented in Figure 4.1. EdgeCloudSim provides a modular architecture to support for a variety of crucial functionalities such as network modeling specific to WLAN and WAN, device mobility model, realistic and tunable load generator. EdgeCloudSim by design supports simulation of multi-tier scenarios

where the multiple edge servers are running in coordination with upper layer cloud solutions. Specific to this need, EdgeCloudSim provides an edge orchestrator module to enable its users to model the task orchestration actions which typically arise in edge computing scenarios.

For modeling the computational tasks such as VM creation with a given capacity, EdgeCloudSim relies on the capabilities of CloudSim, which has a long-known and reliable code base for the simulation of computational actions. EdgeCloudSim is publicly available as an open-source project [9]. Modular design and open-source code base of EdgeCloudSim allow its users to incorporate the specific needs of their scenarios in their simulation experiments.

EdgeCloudSim can be used to analyze different engineering problems in the edge computing domain. The solution of the problems can be applied on different time scales. For example, to improve the service time, various solution methods such as increasing the capacity, changing the offloading decision, and using a more efficient provisioning algorithm can be applied. Here, increasing the capacity can be handled in a broader time scale, since it requires a new deployment. On the other hand, changing the offloading decision will show its effect in a narrower time scale, because it can be done on the edge server dynamically. EdgeCloudSim can be used for evaluating different design decisions in a wide range of time scales. To demonstrate these capabilities of the EdgeCloudSim, we designed three sample simulation setups. In the first setup, different edge computing architectures with different internal organizations are examined. In the second setup, the edge server capacity planning problem is analyzed. Finally, in the third setup, the effect of mobility is explored. These simulation setups are evaluated via an extensive number of experiments carried out with EdgeCloudSim. The results of these experiments clearly depict the effect of the networking, mobility, and architectural parameters on the overall performance of the edge computing systems.

The remainder of this chapter is organized as follows. Section 4.1 addresses some of the challenges in developing an edge computing simulator. In Section 4.2, the

design and the architecture of the EdgeCloudSim are explained. In Section 4.3 and 4.4, example simulation setups are summarized and the results of the experiments are evaluated. Section 4.5 gives discussion and provides the possible directions for future researches.

4.1. Challenges of Developing Edge Computing Simulator

The simulators provide many advantages to the researchers. However, from the simulator developers' perspective, there are some modeling challenges [74]. When compared to the traditional cloud computing simulators, we envision that simulating edge computing environment will bring more complex questions and challenges.

Modeling the virtualized resources (e.g., CPU, memory, storage) and their provisioning is the first challenge to be addressed. Thanks to the cloud computing simulators, we have some current solutions to exploit or get inspired from. However, the applications running on these computing paradigms have different characteristics when considering the clients (brokers). For example, the wireless sensor device applications generally have a Sense-Process-Actuate model, which is beyond allocating a VM on the datacenter. As a result, integrating a computational model of the virtualized resources and fog devices requires more complex application models.

Modeling the mobility is another challenge in this domain. In the cloud computing simulators, the mobility is generally ignored since the main focus is how to provision the virtualized resources with respect to the client requests. However, in the edge computing architectures, mobility of the fog devices, as well as the edge servers, is an essential factor that cannot be ignored. As the devices become more mobile, the system design will be more challenging since the mobility affects many other modules.

The third challenge is *the network modelling*. In edge computing, the interactions among the users, edge nodes, and the cloud may require different access technologies and topologies. For instance, a mobile user can transmit data to the edge node over

a wireless local area network (WLAN), and the edge node forwards the incoming data to the cloud over a WAN. Using a fixed E2E transmission delay in such systems is not reasonable since the number of users is usually considered very large. Therefore, the modeling tool should be capable of employing different delay models to simulate various access technologies such as Wi-Fi, Bluetooth, 4G, and 5G.

The scalability becomes another significant challenge to handle the high number of the entities used in the edge computing architectures. There is a tradeoff between the scalability and the high level of details. If the components are abstracted at a very high level, the tool becomes unrealistic. On the other hand, if too many details are considered, it provides poor scalability.

The extensibility is also an important factor for the researches while choosing a simulator. The researchers usually use different architectures and application models in their studies. As a result, they need to customize and extend the modeling tool according to their requirements. Providing extensible modules is another challenge that should be taken into consideration.

Finally, *the ease of analysis* is a crucial feature required by the simulator users. The results of the simulations can be analyzed regarding various aspects. Determining how many logs will be collected, and the output format of them are the last challenges to be considered while developing an edge computing simulator.

4.2. EdgeCloudSim Architecture

EdgeCloudSim provides a modular architecture where each module addresses a specific aspect of edge computing with clearly defined interfaces to the other modules. Each module contains a default implementation that can be tuned via the simulation parameters to ease fast prototyping efforts. As explained in the previous sections, EdgeCloudSim relies on the capabilities of CloudSim for modeling computational tasks. The VMs served by the hosts are fundamentally the standard VMs provided by CloudSim

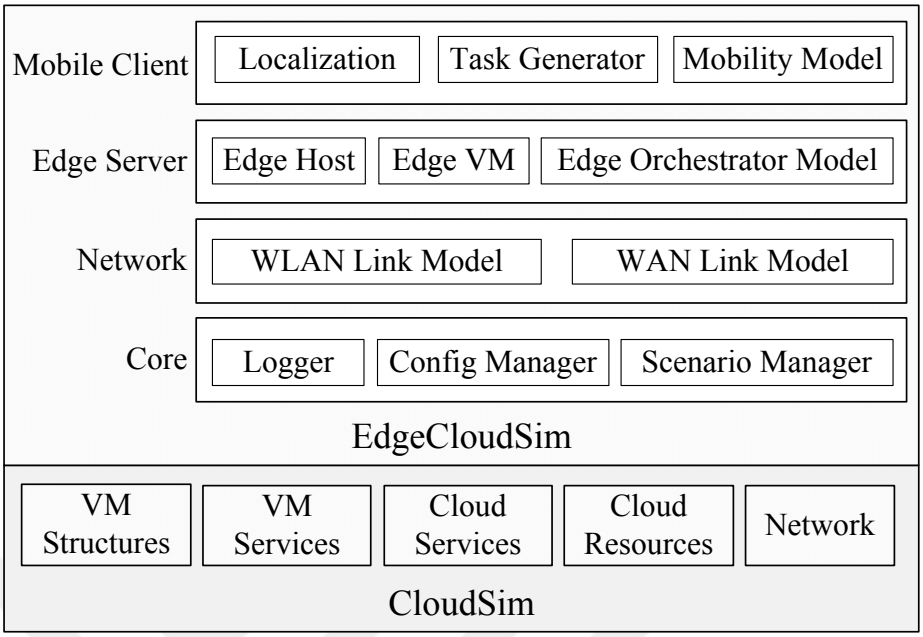


Figure 4.2. EdgeCloudSim block diagram.

with lower capacity and a more realistic CPU utilization model. In CloudSim, the bandwidth, RAM, CPU, and storage resources are limited while creating a VM. When it comes to the tasks, there is no limitation for the number of tasks running on VMs. If there are many tasks to run, simply, the execution of tasks takes a longer time. However, this modeling of execution time is not compatible with the edge computing approach because we expect the edge servers to handle incoming tasks in a short time interval. Therefore, we implement a new CPU utilization model for the VMs where the number of maximum tasks running in parallel on the VMs is limited. The CPU utilization of a single task is static, and it is defined in the configuration file, but a dynamic utilization model can also be implemented by re-writing the related utilization class. A detailed block diagram of EdgeCloudSim modules is shown in Figure 4.2.

4.2.1. EdgeCloudSim Modules

EdgeCloudSim uses CloudSim to provide the cloud computing capabilities such as allocating a VM on datacenters, executing a task on VMs, provisioning the cloud

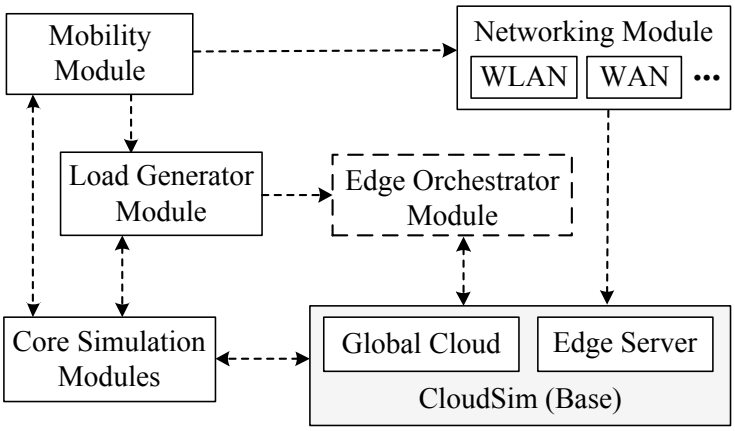


Figure 4.3. Relationship between EdgeCloudSim modules.

resources, and modeling the power consumption of the datacenters. In addition to the cloud computing capabilities, EdgeCloudSim provides other unique modules to handle the edge computing scenarios, as depicted in Figure 4.3. In the current EdgeCloudSim version, there are five main modules available, namely: Core Simulation, Networking, Load Generator, Mobility, and Edge Orchestrator.

The core simulation module is mainly responsible for loading and running the edge computing scenarios from the configuration files. EdgeCloudSim makes it possible to load the properties of edge datacenters, the characteristics of applications, and the basic simulation settings dynamically. Therefore, users can run various simulation scenarios with different configurations without changing their source code. Another essential feature of the core simulation model is the logging feature. Simulation logs are very important to evaluate results. EdgeCloudSim records the results of the simulation in a comma-separated value (CSV) data format. The contents of the file, as well as the format, can also be modified according to the needs.

The edge orchestrator module is designed to fulfill the orchestration process, which is a crucial issue on the edge computing systems. MEO has already been proposed by the ETSI. Similar to the MEO, we propose an edge orchestrator that manages the

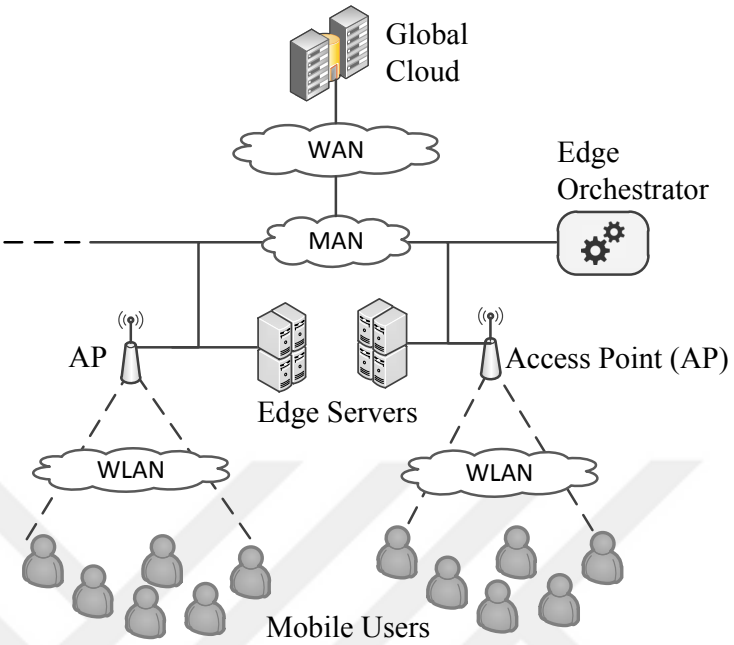


Figure 4.4. Generic edge computing architecture.

resources in a way to increase the overall system performance. The edge orchestrator makes critical decisions such as creating new replicas, terminating the edge VMs, managing the computational resources of hosts, and offloading the tasks to the cloud or edge servers. It should collect information from the other entities to make the decision process more efficient. Therefore, it is closely coupled with the other modules. A generic edge computing architecture is given in Figure 4.4. There are two tiers in this architecture. The mobile devices and the edge services are located in the first tier, and the global cloud services are located in the second tier. In our design, the edge orchestrator is located between these two tiers and works in a centrally controlled manner.

The networking module mainly handles the transmission delay in the WLAN and WAN by considering both upload and download data. The mobile clients may use the cellular network or WLAN to access cloud services. In CloudSim, it is possible to add link delays between the network components, but these delays are static and

fixed for all users. However, the network link quality may vary from time to time if we consider mobile devices. Besides, if many mobile devices are using the same Wi-Fi access point, a high number of devices would inevitably increase the link delay for either the LAN or WLAN communication. In order to model LAN, MAN, and WAN communication more accurately, a networking module is provided in EdgeCloudSim. Before sending a task to a VM or downloading the result of the task from the VM, the transmission delay for the upload/download operation is calculated by the network link model. The default implementation of the networking module is based on a single server queue model. The users of EdgeCloudSim can incorporate their own network behavior models into the networking module. Currently, we are working on a large scale WAN delay characterization crowdsourcing project and plan to incorporate a realistic WAN delay distribution model into the Networking Module.

The mobility module of EdgeCloudSim provides mobility support, which is one of the significant functionalities missing in cloud computing simulators. Since the current simulators focus on conventional cloud computing principles, it is not considered in the framework. However, the devices used in the edge computing environments are mostly mobile. Ignoring the mobility in such systems can lead to incorrect results because the coverage of the wireless networks is limited, and the movement of the users causes problems such as congestion, latency, and packet loss. EdgeCloudSim considers mobility, so the locations of the mobile devices are updated according to the mobility module. In our design, each mobile device has x and y coordinates, which are updated according to the dynamically managed hash table. In addition to the mobile devices, we also define locations where dedicated Wi-Fi access points are utilized. As the default radio behavior, it is assumed that the coverage of the Wi-Fi access points is fixed, and the link quality is the same for all of the covered clients regardless of their distance to the related access point.

The load generator module is responsible for generating tasks for the given application configuration. The mobility and load generator modules are the main components that provide input to other components. They are handled within the same

(Mobile Client) layer, as shown in Figure 4.2. Multiple application types can be defined in EdgeCloudSim, since in most of the cases, servers provide a various number of services. For example, mobile users can offload their tasks corresponding to the face recognition, health monitoring, and augmented reality applications. By default, the tasks are generated according to a Poisson arrival process in EdgeCloudSim. If other load generation models are required, the mobile device manager module should be modified. It should be noted that the data size and the length of the tasks should be decided with a proper distribution with respect to the selected task generation distribution. These variables are exponentially distributed random numbers by default.

4.2.2. Ease of Use and Configurability

The simulators are mostly used to compare the performance of different models by considering many aspects. For example, the topology of the network, the number of entities used in the simulation, or the parameters of the algorithms can change. Handling all these variable options programmatically is a challenging issue. EdgeCloudSim uses three configuration files to decrease the programming effort and provide the configurability. The first file includes the key-value pairs as the simulation parameters such as the simulation time, warm-up period, and the number of mobile devices used in the scenario. The second file includes XML descriptions of the applications used in the simulation and their characteristics such as the task length, the input/output data size, the idle/active period of the applications, the interarrival time of tasks generated by the related application, and the number of instructions to execute the emerging task. The characteristic of the applications is used by the load generator module while creating the tasks. The third file determines the edge server topology and the access points associated with the related location. The location of the edge access points and the specification of the edge devices are defined in this file in XML format. A part of a sample XML file is shown in Figure 4.5. EdgeCloudSim allows the users to run simulations with different configurations by accepting the related files as a command-line option. By doing so, it makes the configuration and execution of the experiments easier.

```
<edge_devices>
    <datacenter arch="x86" os="Linux" vmm="Xen">
        <costPerBw>0.1</costPerBw>
        <costPerSec>3.0</costPerSec>
        <costPerMem>0.05</costPerMem>
        <costPerStorage>0.1</costPerStorage>
        <location>
            <x_pos>1000</x_pos>
            <y_pos>0</y_pos>
            <wlan_id>0</wlan_id>
            <attractiveness>2</attractiveness>
        </location>
        <hosts>
            <host>
                <core>16</core>
                <mips>40000</mips>
                <ram>32000</ram>
                <storage>500000</storage>
                <VMs>
                    <VM vmm="Xen">
                        <core>4</core>
                        <mips>10000</mips>
                        <ram>8000</ram>
                        <storage>125000</storage>
                    </VM>
                    <VM vmm="Xen">
                        <core>4</core>
                        <mips>10000</mips>
                        ...
```

Figure 4.5. Sample XML data for edge server configuration.

4.2.3. Extensibility

Extensibility is one of the essential features of the software tools. It provides a customizable environment in which the developers can change the behavior of components without changing the existing codebase. If the extensibility is low, the developers do not tend to use the related tool because they have to learn the details of existing architecture for customization and enhancement. EdgeCloudSim provides the default implementation of the modules mentioned earlier in this section. For example, it includes the nomadic mobility model as the mobility module and the simple queue model as the networking module. However, users may want to implement these modules differently depending on their needs. To prevent modifying the existing codebase, Edge-

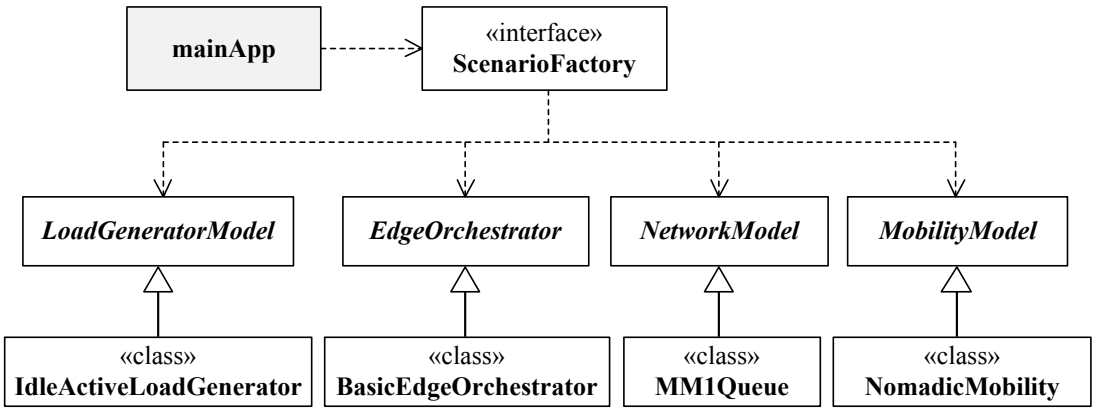


Figure 4.6. Factory pattern for simulation scenario.

CloudSim uses a factory pattern making it easier to integrate the new modules. As shown in Figure 4.6, the scenario factory class knows the creation logic of the abstract modules and provides the concrete implementation of them. As a result, the users can utilize their own modules in EdgeCloudSim by taking advantage of this pattern. The scenario factory is an abstract class and provides four different modules, which are the mobility, load generator, networking, and the edge orchestrator modules. This abstract class can be extended to support the modeling of custom modules.

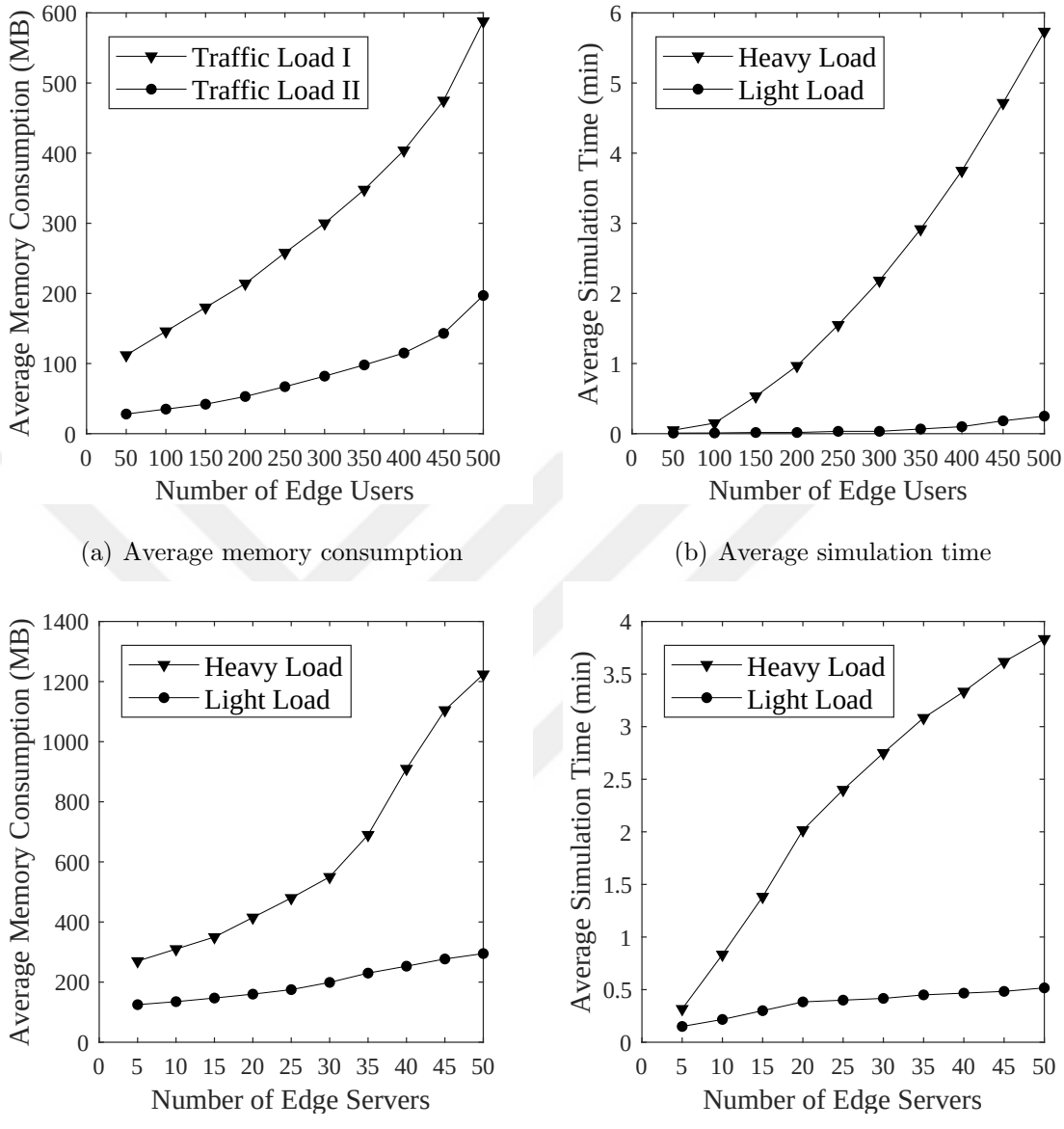
4.2.4. Complexity Evaluation

EdgeCloudSim runs on top of CloudSim which uses a discrete event simulation (DES) [32] engine. Therefore, the complexity of EdgeCloudSim depends on the number of events created in the simulation and the algorithms used in the modules. For example, if the designed edge orchestrator is too complex, the memory consumption is more likely to be high, and the duration of the simulation is long. In our study, an experiment is setup to present how much memory does EdgeCloudsim use and how much time does it take to execute an ordinary simulation. A basic personal computer running on a Linux operating system with Intel Core i7-5600U processor and 16 GB memory is used in the experiment. One hour-long real-life events are simulated under light and heavy load scenarios. The edge computing architecture used in the experi-

ments is called *two-tier with edge orchestrator*, which consists of mobile devices, edge servers, and cloud servers. The mobile devices operate three different applications, which correspond to augmented reality, infotainment, and health monitoring applications. The modeling of the applications, mobility, network, and the two-tier with edge orchestrator architecture are explained in Section 4.3.

In the experiments, the time and memory complexities are analyzed for two different cases. In the first case, the memory consumption and the duration of the simulation processes are investigated with respect to the variable mobile devices. In the heavy load scenario, the augmented reality, infotainment, and health monitoring applications generate task offloading requests every 2, 5, and 10 seconds on average, whereas, for the light load scenario, the average interarrival time of tasks are 10, 25, and 50 seconds. A considerable number of events are generated during the simulations, especially for the heavy load scenario. The results are shown in Figure 4.7(a) and 4.7(b). It can be observed from the figures that even a quite dense scenario with a heavy load can be completed in a couple of minutes with half GB memory on a standard portable computer.

In the second case, the time and memory complexities are investigated with respect to the variable number of edge servers. The edge datacenters include host machines and VMs. In this experiment, each edge server has one host which operates 2 VMs with 10 Giga instruction per second (GIPS) CPU power. In the heavy load scenario, 500 mobile devices are used, whereas there are 200 mobile devices in the light load scenario. The memory consumption and the duration of the simulation process are shown in Figure 4.7(c) and 4.7(d). It can be considered that if the number of edge servers is high, the geographical area of the simulation scenario is also large. According to our evaluation, EdgeCloudSim can execute relatively large instances with 500 mobile devices and 50 edge servers operating 100 edge VMs in a reasonable time interval with acceptable memory consumption.



(a) Average memory consumption

(b) Average simulation time

(c) Average memory consumption

(d) Average simulation time

Figure 4.7. Complexity analysis of EdgeCloudSim for heavy and light load scenarios.

4.3. EdgeCloudSim in Operation

In order to present EdgeCloudSim capabilities, three simulation setups are designed, and the results are evaluated. In these setups, the effect of edge computing architecture, edge server capacity, and the mobility on the important performance metrics are investigated. We design a virtual environment that is similar to a university campus in our experiments. The students walk around and request services from the edge servers that are located at the buildings on the specific territories. Each building

is also serving a wireless access point, and the mobile users are connected to the related access point and offload their tasks via this connection. A step-by-step explanation of how the EdgeCloudSim components are modeled to obtain an evaluation as close as possible to the real-world scenarios is described in detail later in this section.

4.3.1. Step 1: Modeling the Applications

In real life, the applications running on the mobile devices of the students can be variable. For example, it can be an augmented reality application on smart glass [75], an infotainment application [76], or a health application using a foot-mounted inertial sensor [77]. In our simulations, the mobile devices utilize three different applications, and the edge and cloud servers provide corresponding services. The load generator module of EdgeCloudSim is used to characterize the tasks generated by the applications mentioned above. The configuration of these applications is listed in Table 4.1.

Typically, mobile devices do not generate service requests continuously. We use an idle/active task generation pattern to simulate real-life accurately. According to this pattern, the users create tasks during the active period, and then they wait during the

Table 4.1. Applications used in the simulations.

	Augmented Reality	**Infotainment App**	**Health Monitoring**
Usage Percentage (%)	50	30	20
Task Interarrival Time (sec)	2	5	10
Idle Period Duration (sec)	20	25	90
Active Period Duration (sec)	40	45	15
Upload Data Size (KB)	1500	25	1250
Download Data Size (KB)	25	750	250
Task Length (GI)	20	7.5	2.5
VM Utilization of Tasks (%)	10	5	2

idle period. It can be considered that there is a state machine for each client where the client can be either in the active or idle state. For example, the current wearable devices support both the heart rate and physical activity monitoring features. The gathered information on the wearable device can be sent to the nearby device to apply more sophisticated data processing. In these circumstances, the wearable device collects data for some time, and then it sends it to the edge services.

The tasks generated by the applications have random lengths in terms of the number of instructions and random input/output file sizes for upload/download operations. For example, the augmented reality application requires high CPU computation, a small amount of data to download, and a larger amount of data to upload. Therefore, we use an average of 1500 KB upload-data size considering a typical jpeg image and 15 KB download-data size referring to the recognized person's metadata. On the other hand, the infotainment application requires less CPU power compared to the augmented reality application. It has a request with a small amount of data, and the corresponding service returns a larger amount of data as a response. The task length directly affects the execution time of the task offloaded to the edge servers depending on the computational power of the VM running on the related server. In our simulations, the computational resources of cloud entities are assumed to be sufficiently large for all scenarios. Therefore, it can be assumed that each task offloaded to the cloud server is executed on a VM without blocking. However, congestion may occur in edge servers.

4.3.2. Step 2: Modeling the Mobility

The mobility module of EdgeCloudSim is used to model the mobility of mobile users. In our experiments, the movement of the users is modeled based on a nomadic mobility model [78]. They wait in one place for a random amount of time and then move to another place. There are three location types with different attractiveness levels in our model. If the attractiveness level of the location is high, the user is likely to spend more time in that place. For example, the users spend more time in a library

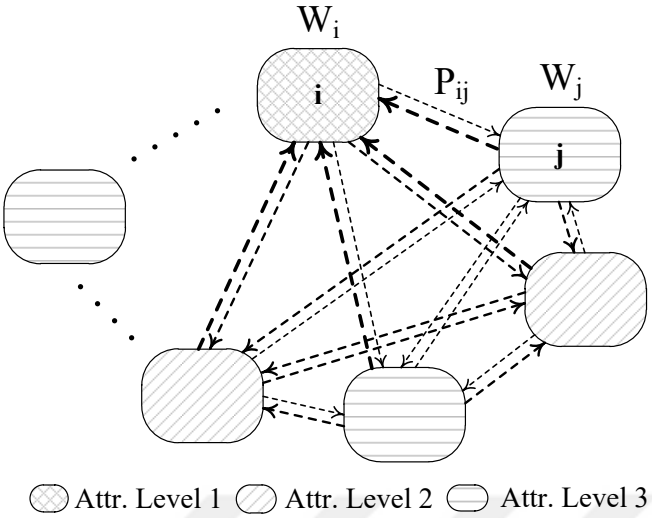


Figure 4.8. Mobility model used in the simulations.

or a cafe than a kiosk. The mobility model is illustrated in Figure 4.8, where W_i is the dwell time of the user at the location i, and P_{ij} refers to the probability of moving from location i to j. In our simulation, the attractiveness level of the location directly affects the dwell time that the user spends in the related place. The probability of moving to any location is the same for all locations.

4.3.3. Step 3: Modeling the Network Delay

The networking module of EdgeCloudSim is used to calculate the network delay. In our experiments, each place in the proposed environment is covered by a dedicated Wi-Fi access point. When the mobile devices move to the coverage area of the access point, they join the related WLAN. Then the mobile devices start sending tasks to the edge server. If a task is decided to be offloaded to the global cloud, the WAN connection provided by the Wi-Fi access point is used. EdgeCloudSim provides a default simple queue model as the networking module for WLAN and WAN models. To achieve a more realistic simulation environment, we use an empirical WLAN and WAN model by extending the default networking module. Another network model can also be used if a different access technology model. The experimental network model is developed

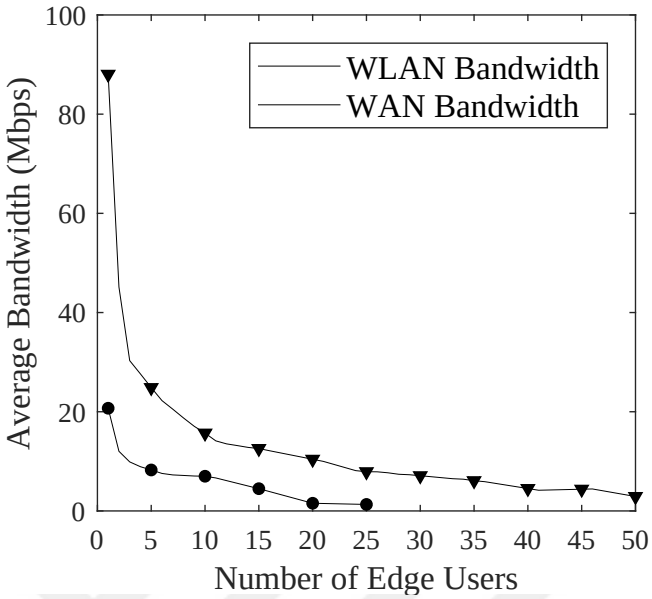


Figure 4.9. Measured delay values used in the empirical WAN and WLAN capacity models in the simulator.

by taking measurements from real-life deployments. For WLAN delay observation, an 802.11 family access point is used, and a fiber internet connection in Istanbul was utilized to observe WAN delays. The results of the empirical network delay analysis are shown in Figure 4.9.

4.3.4. Step 4: Modeling the Edge Computing Architectures

In order to demonstrate the capabilities of EdgeCloudSim, we experimented with three simulation setups. In the first setup, we compared three architectures, which are (i) single-tier, (ii) two-tier, and (iii) two-tier with edge orchestrator (EO). As shown in Figure 4.4, the generic edge computing architecture includes both edge servers and the cloud servers. The single-tier architecture allows the mobile devices to utilize the edge server located in the same building with the student. When it comes to the two-tier architecture, the mobile devices can send their tasks to the global cloud by using the WAN connection provided by the connected access point. In real-life scenarios, the decision to select which service to utilize is decided by the edge orchestrator. In our

Table 4.2. Simulation parameters.

Parameter	**Value**
Simulation time	33 minutes
Warm-up period	3 minutes
Number of repetitions	100
Number of places for type 1/2/3	2/4/8
Mean waiting time in place type 1/2/3	8/4/2 minutes
Provisioning Algorithm on Edge	Least-Loaded
Probability of Cloud Offloading	0.1
Number of VMs per edge server	8
CPU speed per edge VM	10 GIPS
CPU speed per cloud VM	200 GIPS
LAN propagation delay	5 milliseconds

simulation, the offloading ratio to the cloud is decided as 10 percent. Hence approximately one out of ten tasks is offloaded to the cloud. The two-tier with EO architecture has a considerable advantage, because, for the tasks which are executed on the first tier, only the two-tier with EO architecture can offload the tasks to other edge servers located in different buildings. In this simulation setup, the edge orchestrator uses the least-loaded algorithm while selecting an edge server to offload in the first tier. It is assumed that the edge servers and the edge orchestrator are connected to the campus network so that they can exchange information in a fast manner. Other important parameters used in the first simulation setup are listed in Table 4.2.

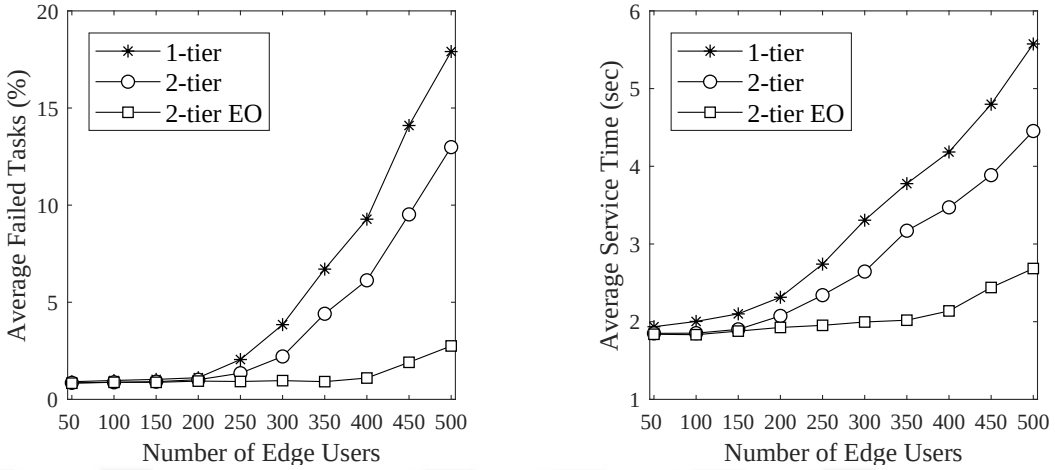
In the second simulation setup, the effect of the edge server capacity on the results has been investigated. Deciding the edge servers' capacity is a significant problem that should be handled before the deployment of the edge system. In this setup, we evaluate three different deployment scenarios by the same parameters given in Table 4.2. We use 80, 40, and 20 GIPS edge servers for the first, second, and third deployments, respectively. Since the CPU speed of VMs is 10 GIPS, the number of VMs per edge

server becomes 8, 4, and 2 for the related deployments.

In the third simulation setup, the impact of the dwell time of the students on the results was investigated. Since we use a nomadic mobility model, the location of the students is changed after a random amount of time passed. The dwell time of the students is calculated based on the model given in Figure 4.8. In this setup, we use three scenarios with different dwell time options for each location type. In the first scenario, which is called *fast*, the mean waiting times in the places with attractiveness levels 1, 2, and 3 are 0.5, 1, and 2 minutes respectively. In the second scenario, which is called *normal*, the mean waiting times in the related places are 2, 4, and 8 minutes. Finally, in the third scenario, which is called *slow*, the mean waiting times in the related places are 6, 12, and 24 minutes. The rest of the simulation parameters are the same as in Table 4.2.

4.4. Simulation Results

As already mentioned, three simulation setups are investigated in order to present EdgeCloudSim capabilities. The first setup evaluates the performance of different edge computing architectures. Important performance metrics are shown in Figure 4.10. In Figure 4.10(a), the average task failure values with respect to the number of mobile devices are given. In Figure 4.10(b), the average service time is shown. The devices in the single-tier architecture can only offload to the nearest edge server. As a result, the devices experience congestion at attractive places with a relatively higher number of users. Therefore, the number of failed tasks in single-tier architecture is observed higher than the other architectures. Similarly, the average service time is worse than its competitor due to the congestion that occurred on the hot spot locations. When it comes to the two-tier architectures, they present better results than the single-tier architecture because some of the tasks can be offloaded to the cloud servers. In this work, a basic probabilistic approach is used to decide offloading tasks to the cloud. Roughly, 10 percent of the tasks are sent to the cloud, so the two-tier architecture is slightly better than the single-tier. If s more sophisticated offloading



(a) Average Failed Tasks

(b) Average Service Time

Figure 4.10. Comparative evaluation of the different edge computing architectures based on all application types.

mechanism is designed, the performance difference will be more than the presented scores. The two-tier with edge orchestrator architecture outperforms the others since the edge orchestrator makes it possible to send the tasks to any edge server in the same LAN. Therefore, it can balance the load of the edge servers efficiently and avoids the congestion occurred in attractive places where too many users are present.

In Figure 4.11, the average WLAN delay values with respect to the number of mobile devices are shown. According to our experimental network model, the delay between the mobile devices and the edge servers becomes higher as the number of users connected to the same WLAN increases. Compared to the single-tier architecture, the two-tier architectures provide better WLAN delay performance, because they can redirect some of the requests to the global cloud. The LAN delay in the two-tier with an edge orchestrator architecture is growing faster than the two-tier architecture. It is misleading to evaluate this graph alone since the results are also related to task losses. The average task failure is very low in the two-tier with edge orchestrator architecture. Thus the number of mobile devices sending/receiving tasks simultaneously is getting higher. As a result, in this architecture, the WLAN delay increases faster as the number of mobile devices increases.

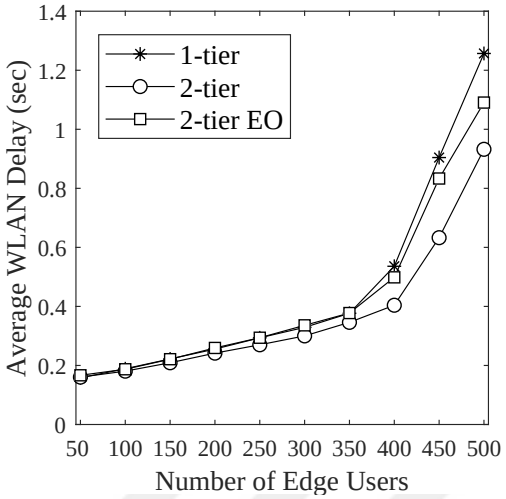
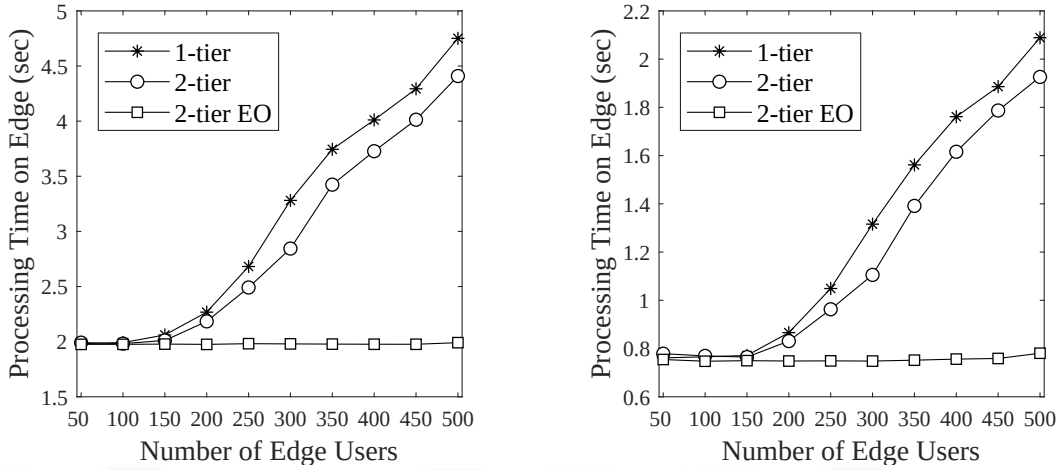


Figure 4.11. Average WLAN Delay based on all application types.

The results given in Figure 4.10 are the average scores of all applications which generate tasks to be executed on the edge or cloud servers. EdgeCloudSim can provide the scores of each application separately as well. For example, the average processing time of different applications on the edge servers is shown in Figure 4.12. The processing time is the duration spent on the server while executing a task request. Since EdgeCloudSim runs on top of CloudSim, the processing time is calculated by the core CloudSim modules. Essentially, the processing time becomes higher if the number of the tasks running on the server increases, because the hosts used in our simulations operate a *space shared* VM scheduler. The average processing time of the tasks generated by the augmented reality and infotainment applications are shown in Figure 4.12(a) and Figure 4.12(b) respectively. Both figures provide similar slopes for each architecture, but the values are different. This is an expected result since the task length of the augmented reality application is higher than the infotainment application. The average processing time performance of the architectures are similar to the average service time performance in Figure 4.10(b). The two-tier with an edge orchestrator architecture outperforms the others because it can distribute tasks among the edge servers. The two-tier architecture provides slightly better performance than the single-tier architecture since it can relay some tasks to the cloud servers.



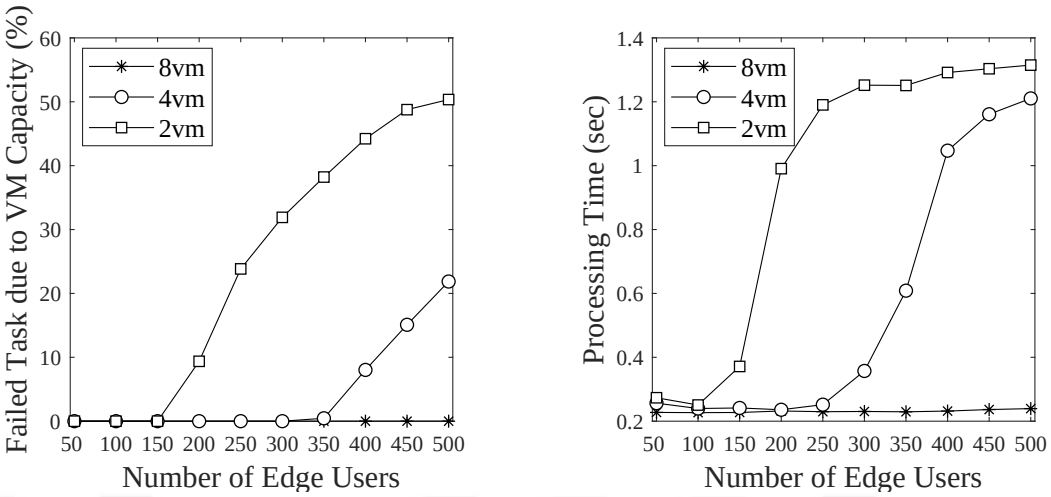
(a) Average processing time for augmented reality application

(b) Average processing time for infotainment application

Figure 4.12. Average processing time for different application types.

In the second simulation setup, the effect of the edge server capacity on the results is investigated. In Figure 4.13, the performance of three deployments are compared. The deployments include different edge servers in terms of the CPU power. The most powerful edge server can operate 8 VMs, the least powerful one can operate 2 VMs, and the other one can operate 4 VMs. The average failed tasks due to the VM capacity and the average processing time values are shown in Figure 4.13(a) and Figure 4.13(b) respectively. The edge servers with 2 VMs start to experience congestion after 150 mobile devices, and the congestion starts after 350 devices for the edge servers with 4 VMs. The system can handle even 500 mobile devices if the edge servers operate 8 VMs. These results are not surprising; in fact, they can be estimated. The main motivation for investigating such a simulation setup is presenting how EdgeCloudSim can be used to evaluate and optimize the edge server capacity planning.

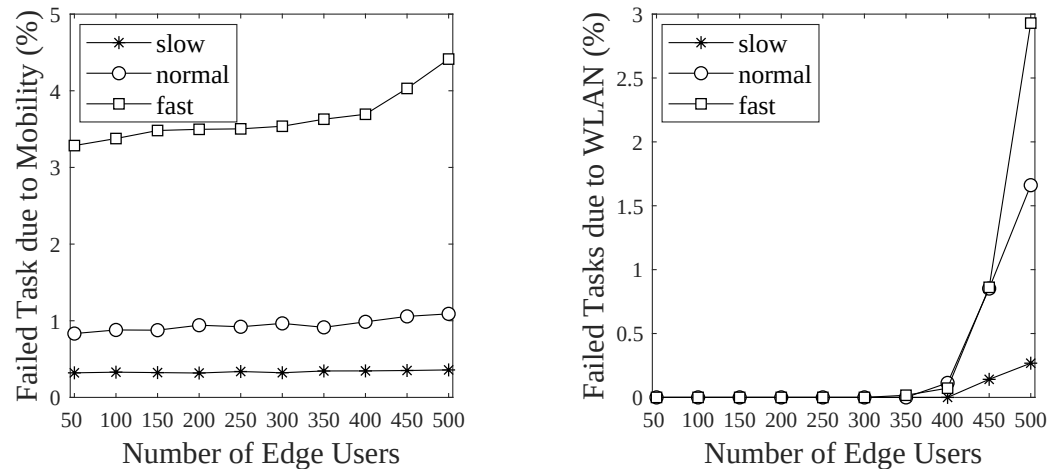
In the last simulation setup, the effect of the mobility on the simulation results is investigated. The nomadic mobility model is used in our simulations; hence the speed of the mobile devices can be configured by modifying the dwell time parameter. In this setup, slow, normal, and fast scenarios are evaluated where the dwell time of the users for each place time is modified accordingly. The average failed tasks due to the



(a) Avg. failed task due to VM capacity (b) Average processing time

Figure 4.13. Performance evaluation of the system for different VM sizes.

mobility is given in Figure 4.14(a). The task failure due to mobility occurs when the mobile device offloads a task and leaves the coverage area of the WLAN before getting the response. The task failure due to mobility occurs when the mobile device offloads a task and leaves the coverage area of the WLAN before getting the response. It can be clearly seen that the number of tasks which are failed due to the mobility increases in parallel to the residence time of the users. The average failed tasks due to the WLAN



(a) Avg. failed task due to VM mobility (b) Avg. failed task due to WLAN failure

Figure 4.14. Performance evaluation of the system for different dwell time values.

failure are shown in Figure 4.14(b). If the users are too fast, the congestion occurs more likely in the hot spot places. As a result, the task failure ratio due to the WLAN congestion is proportional to the speed of the users.

4.5. Discussion

The cost of experimenting on real edge/cloud environments or testbeds is a factor that pushes the researchers to evaluate their proposals through the use of simulators. EdgeCloudSim addresses this need for the edge computing domain, which depends on both computational and networking aspects. EdgeCloudSim provides the mobility model, network link model, and edge server model to evaluate the various facets of edge computing. In addition to the simulational capabilities, EdgeCloudSim depicts increased usability by providing a mechanism to get the configuration of devices and applications from the XML files instead of defining them programmatically. We provide this open edge computing simulator publicly available on GitHub to enable and motivate the researches on the edge computing area [73]. As a future plan, we plan to add a hand-off mechanism to EdgeCloudSim. The hand-off mechanism would decrease task failures due to mobility by using a VM migration method. Another future work we planned is adding the mist computing [79] feature to enhance the computational capabilities of the mobile devices. Finally, we want to add an energy consumption model for the mobile and edge devices as well as the cloud datacenters.

5. FUZZY WORKLOAD ORCHESTRATION FOR EDGE COMPUTING

We are witnessing a tremendous change in the end-user side that generates traffic for today's networks. The proliferation of the commercially available mobile gadgets, sensors, wearable units rendered many novel applications feasible involving IoT, smart home/city, autonomous vehicles, augmented reality, instant video processing, tactile internet, e-health, and Industry 4.0. These applications dictate a totally different modality of the workflow when compared with traditional client-server applications served over a cloud computing infrastructure: they create vast amounts of data, require a prompt response (real-time or semi real-time), involve high user mobility and diverse spectrum of requirements.

Although this new category of applications are demanding in terms of the expectation of its users, the equipments employed for realizing them are getting even smaller, resulting in stringent battery and computing limits. This, in essence, presents a new challenge where the complex software is being implemented on constrained hardware. Edge computing addresses this challenge by boosting the end-user device via computational offloading mechanisms [1] through the use of micro-cloud servers located closer to the end user.

Edge computing is actually an umbrella term that signifies a family of relevant technologies such as cloudlet based computing [2], MCC [3], Fog Computing [4] and MEC [5]. All these approaches are based on the sound engineering principle that computational resources are better be located on the edge where the real demand is physically present. By doing so (i) the end-user devices can get fast access to local computational resources using the LAN (ii) the core network is not kept under heavy traffic load (especially critical for 5G), i.e the end-user devices' data can be processed before sending it to the cloud.

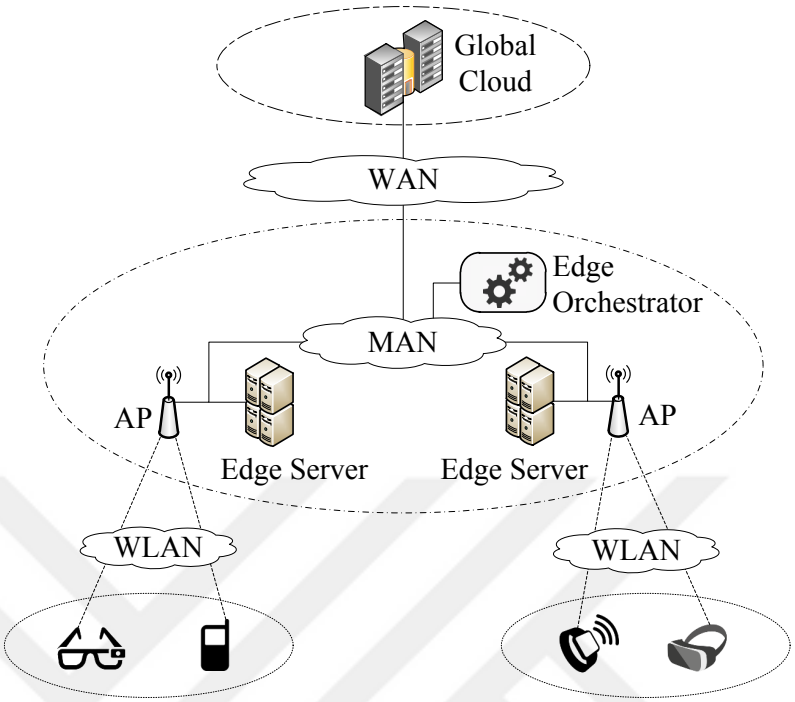


Figure 5.1. Multi-tier edge computing architecture with edge orchestrator.

Edge computing infrastructure involves a series of edge servers located in a metropolitan area, connected with a high-speed network and managed from a single administrative domain. Cloud computing resources are also a natural part of this infrastructure, aiding to serve whenever edge resources are insufficient. We depict this multi-tier view of the system in Figure 5.1. In this picture, offloaded tasks from the end-users can be served on any edge server in the domain based on the availability of the service requested and the spare capacity of the servers.

Envisioned edge computing scenarios constitute a very dynamic overall operation being executed on a fairly elastic infrastructure with (i) many devices and gadgets getting in and out of the local edge coverage, (ii) a diverse set of applications resulting in a stream of tasks with varying complexity, (iii) networking state instantaneously changing and (iv) fluctuating computational capacity which is virtualized and dispersed over the local edge and cloud servers [80, 81].

In this heterogeneous environment, how the edge system responds to the incoming stream of offloaded tasks determines the overall efficiency and scalability of the system. Moreover, the satisfaction of the end-user depends on whether tasks are handled in accordance with the quality expectations. With this vision in mind, one of the significant challenges of the edge computing systems is designing its operation so that a dynamic flow of requests can be efficiently processed. Our work focuses on the workload orchestration problem, which can be defined as deciding on the destination computational unit for each offloaded task within an edge computing system. We model the edge computing system as a multi-tier organization which incorporates virtualized computational resources at various levels, as well as, networking capabilities at different capacities, such as LAN, metropolitan area network (MAN) and WAN. In this respect, the workload orchestration problem should take into account the states of both computational and networking resources at hand.

In this study, we propose Fuzzy Logic as the tool to solve the workload orchestration problem in the edge computing systems. Fuzzy logic is among the most employed methods to cope with the rapidly changing uncertain systems where it is difficult to provide accurate mathematical models [82]. In addition, the computational complexity of the fuzzy logic-based approaches is low compared to other decision-making algorithms [83]. Being a computationally efficient solution is one of the most significant requirements of the online and real-time problems. Furthermore, Fuzzy Logic is based on well-understood principles and provides a compelling enough method for stating intricate operational details in a high level human understandable format. Our approach uses fuzzy rules to define the necessary workload orchestration actions in terms of networking, computation, and task requirements at hand to decide the task execution location within the overall edge computing system. This enables administrators or end-users to be exempt from identifying operational states and assigning procedures related to these states. Our work resembles intent-based solutions proposed in the literature [84] in the sense that the parameters that are relevant to the operation of the whole system are expressed as rules in a high-level description.

We evaluated the performance of our system through a series of detailed simulation experiments. To achieve this, we realistically modeled a multi-tier edge computing infrastructure using EdgeCloudSim [73]. EdgeCloudSim is capable of modeling the networking as well as the computational aspects of the edge and cloud servers at the VM resolution. Similarly, in our experiments, the dynamic behavior of LAN, MAN, and WAN networking related resources are incorporated.

To realistically model the incoming task traffic, four application classes are considered, namely Augmented Reality, Heath, Compute Intensive, and Infotainment applications. Each application class represents a different profile in terms of task arrival distribution, delay tolerance, and task length. Our method is comparatively evaluated with four opponent algorithms using crucial performance metrics such as service time, failed tasks, and VM utilization. According to the results, our proposed method has competitive performance with respect to its opponents for the cases studied.

The remainder of this chapter is organized as follows. Section 5.1 puts forward the multi-tier edge computing architecture that we employ in our study. Section 5.2 presents a brief explanation of workload orchestration. Section 5.3 describes the proposed fuzzy logic based workload orchestrator. Section 5.4 elaborates on the simulation experiment design and analyzes the results.

5.1. Multi-Tier Edge Computing Architecture

In the early proposals of edge computing, the interaction between the user device and a single edge server was of primary concern [2]. However, as edge computing moves toward becoming a mainstream paradigm, a complete view of the overall system, including both edge and cloud servers, has emerged. As the edge services are envisioned to operate in coordination with the cloud tier as well, a multi-tier architecture covering both types of components becomes a necessity. In that respect, we incorporate multiple tiers in our edge computing architecture as previously depicted in Figure 5.1.

There are mainly three tiers available in our architecture. The mobile devices themselves represent the first tier where they make use of their wireless transmission interference to interact with WLAN gateways. In the second tier, both the edge servers and the edge orchestrator are located. The edge orchestrator is aware of the status of the edge servers as well as the network resources which are used for workload orchestration. Finally, the third tier consists of the global cloud servers, which are typically provided by the cloud service providers such as Amazon EC2 [39], Microsoft Azure [40], and Google Cloud Platform [85].

To clearly demonstrate how our proposed approach fits into a real-world implementation, we modeled the experimentation phase with a focus on the university campus scenario. According to this model, the students and other participants of typical campus demand services from the edge servers within the campus while moving around the predesignated locations such as the classroom buildings, cafeteria, dormitories, and administrative offices. The individuals are assumed to carry and/or wear mobile gadgets, which continuously run applications that create traffic due to the task offloading. There exists a certain number of edge servers located in the campus such that they provide coverage for edge computing requests. A user is connected to the nearest edge server via WLAN, whereas the edge servers are interconnected by MAN. Also, the standard cloud infrastructure can be accessed over WAN. In this setting, it is possible for the user to offload a task to (i) Nearest edge server (we call this the local edge server) (ii) a neighboring edge server within the campus connected by MAN (we call this the remote edge server), and (iii) to the global cloud servers. In our work, this decision is handled by the workload orchestrator.

To realistically model the mobility of the users, the locations on the campus are assumed to have different attractiveness levels. The attractiveness level affects the dwell time of the related place. For example, the students may spend more time in a library or a cafe compared to the other locations. The user density, hence the requested traffic volume, at each place varies depending on the attractiveness level of the place. We use three location categories with different attractiveness levels, as shown in Figure

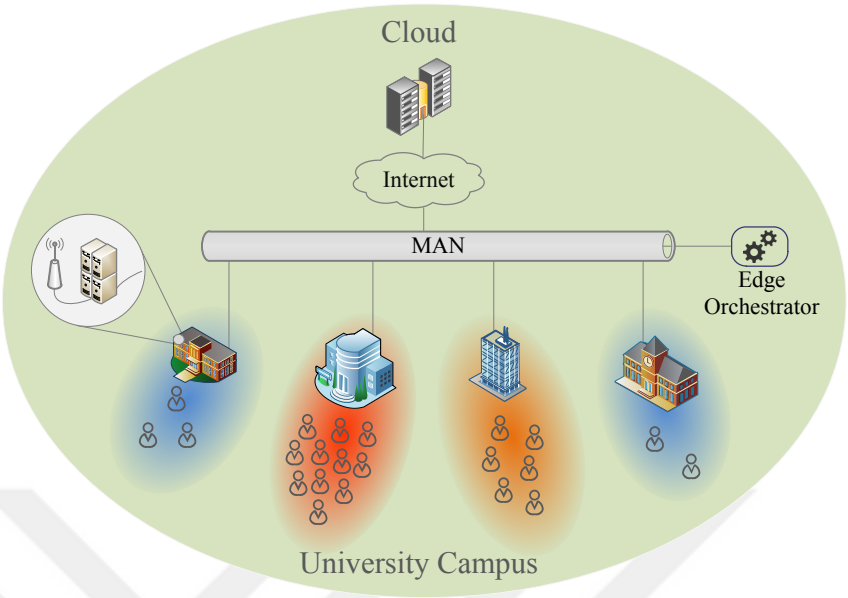


Figure 5.2. Proposed scenario representing multiple locations of a campus distributed in a university.

5.2. A *Type 1* location is shown in a red ellipse, and it has the highest attractiveness level. On the other hand, a *Type 3* location is shown in a blue ellipse, and it has the lowest attractiveness level. When it comes to the *Type 2* location, it has the medium attractiveness level and shown in an orange ellipse. The mobile device density affects the overall system performance due to its direct relation with the requested traffic size. For example, both the network and computational resources are likely to be congested in the *Type 1* location because of the high device density.

5.2. Workload Orchestration

The orchestration is critical for the efficient and robust operation of the multi-tier edge computing systems. With this need in mind, MEO has been already defined by the ETSI [5]. According to ETSI's proposal, MEO knows the topology of the network (the deployed hosts and the mobile devices), the services provided by the hosts, available resources on each host, and the applications used by the mobile devices. MEO works in a centrally controlled manner, and it is mainly responsible for selecting the appropriate

mobile edge host(s) based on constraints, such as latency, available resources, and available services [86].

We focus on an essential component of the overall orchestration effort, namely the workload orchestration in our work. The workload orchestrator determines the target computational unit against an incoming request by monitoring multiple factors such as the network bandwidth, edge server utilization, and task characteristics. In our architecture, the workload orchestrator receives the offloaded tasks from the user devices. The target server is decided via the two-stage decision-making process. As the first step, the workload orchestrator finds a favorable server in the edge layer. Then, it determines whether the task should be executed on the corresponding edge server or the cloud. If the task is decided to be executed on the edge server, the VM with the lowest CPU utilization is selected. The same rule is also valid for the cloud layer. Therefore, the VM having the lowest CPU utilization is used in the cloud datacenter.

There exist many system parameters that have an impact on this decision. In Section 5.3, we explain how this problem can be solved using fuzzy logic. The task flow in the proposed multi-tier edge computing architecture is illustrated in Figure 5.3.

In order to execute a task on the servers, some requirements should be met. Firstly, there should be enough capacity on the network for successful offloading. Secondly, the tasks increase the utilization of the servers based on their complexity. Task complexity is chosen from a distribution characterizing the application profile. There should be enough capacity on the server to execute the related task on it. When it comes to the task length, it is not a conditional requirement. In fact, it directly affects the execution time of the task. Different task length distributions are selected to realistically model the incoming task traffic to the proposed edge system. The dependency between the tasks is not addressed in this study. Therefore, the tasks are considered as stateless jobs. However, a separate framework that handles the task dependency can easily utilize our proposed solution. Since the workload orchestrator contains centralized information, the tasks tagged as a dependent could be executed on the same edge

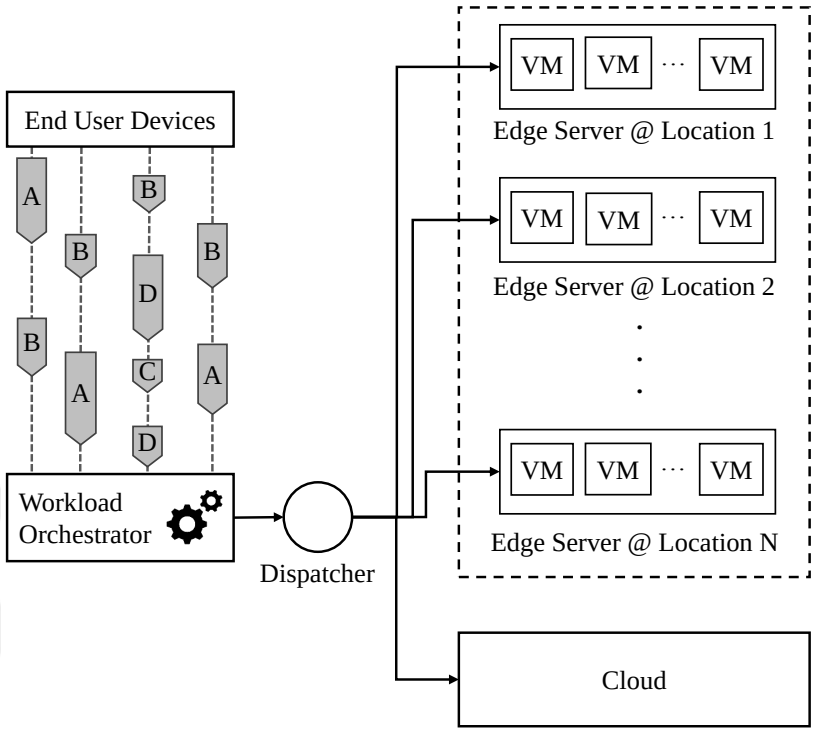


Figure 5.3. Workload orchestration flow.

server to eliminate the extra communication overhead.

In order to design an efficient edge orchestrator, a sufficient amount of information will be needed. However, the data transmission frequency between the edge orchestrator and the other devices should be carefully designed in order not to cause a communication overhead. Therefore, the location of the edge orchestrator becomes a crucial design decision. Practically, the edge orchestrator can be deployed in any tier in the topology, independent of the technology, but dependent on the business model. It can be located on the cloud, edge, or directly on the mobile devices in a distributed manner. For example, Imagane et al. deploy the edge orchestrator on the edge layer for dynamic scaling of the computational and network resources [87]. They use OpenStack to construct the edge computing platforms in their university. Each edge computing platform also includes a regional controller node, and the orchestrator connects to each controller node via an edge network. Our work assumes WLAN for

edge access. However, the proposed system can equally be applied to telecom based networks with cellular access technologies. This would result in a different delay model. Also, the orchestrator location should then be chosen in accordance with the requirements and constraints of the mobile operator. For instance, operating the multi-access edge orchestrator on the Cloud Radio Access Network (Cloud RAN) in next-generation 5G networks could be an alternative.

5.3. Fuzzy Logic Based Workload Orchestrator

For the effective workload orchestration of a multi-tier edge computing environment, the main factors that should be considered are the utilization level of the servers and the network conditions on the routes to these servers. These parameters can be highly dynamic, especially under unexpected variation of the load. For instance, as the number of users increases in a shared network, the bandwidth fluctuations are inevitable. Similarly, the CPU utilization of a VM can exhibit frequent change depending on the tasks running on it. This uncertainty makes it difficult to decide where the offloaded task should run. A realistic model of the edge computing system involves many parameters coming not only from the edge infrastructure itself but also from the end-user device and application characteristics, which further complicates the workload orchestration problem. Multi-constraint optimization with a mathematical model is not an easy task, and generally, a robust model cannot be found due to the complexity of the problem.

Moreover, the conventional offline optimization techniques cannot be applied for the orchestration problem since the offloading requests are not known in advance. Due to the incomplete information on the arrival times and sizes of the tasks, the problem dictates an online solution. Fuzzy logic, in this respect, is an excellent alternative to handle real-world imprecision in the area of computer networking [88]. It has been successfully applied to many networking related areas such as load management, network management, buffer management, and congestion mitigation. We employ fuzzy logic in our workload orchestrator design for a multitude of reasons. Firstly, it can handle the

uncertainty in unpredictable environments without the need for detailed mathematical models. In addition, fuzzy logic easily handles the multi-criteria decision-making process by allowing the consideration of multiple parameters in the same framework [89]. Secondly, it has low computational complexity, which is an essential criterion for an online algorithm. Thirdly, it provides an intuitive interface for system operators since fuzzy logic mimics expert judgment by modeling through the use of imprecise information provided in the form of human language. Finally, it enhances the performance of the system significantly, as demonstrated in the result section.

5.3.1. Two Stage Fuzzy Workload Orchestration

The proposed fuzzy logic based workload orchestrator finds a target server that can be the local edge server, remote edge server, or the cloud server. Handling all of these servers in a single fuzzy logic system (FLS) is a complex operation. Therefore, we use a two-stage FLS to decrease this complexity, as shown in Figure 5.4. In the first stage, the most convenient edge server in the edge layer is found. In the second stage, the candidate edge server and the cloud server are compared. As a result of these operations, the target edge or cloud server is determined.

FLS has particularly used for mapping uncertain variables from the complex

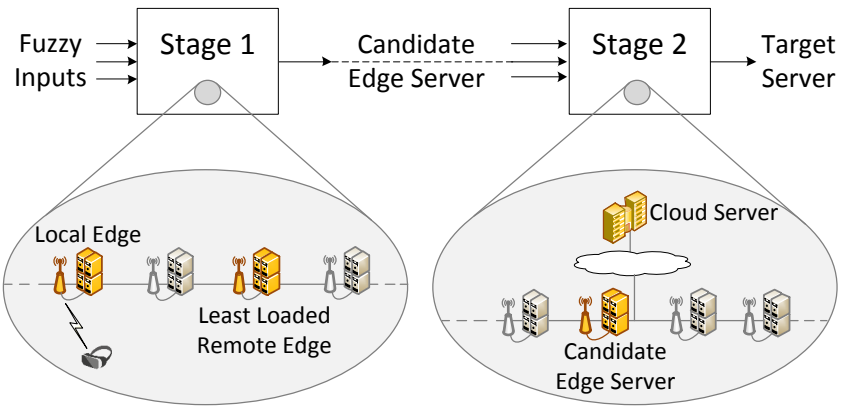


Figure 5.4. Two stage fuzzy workload orchestration.

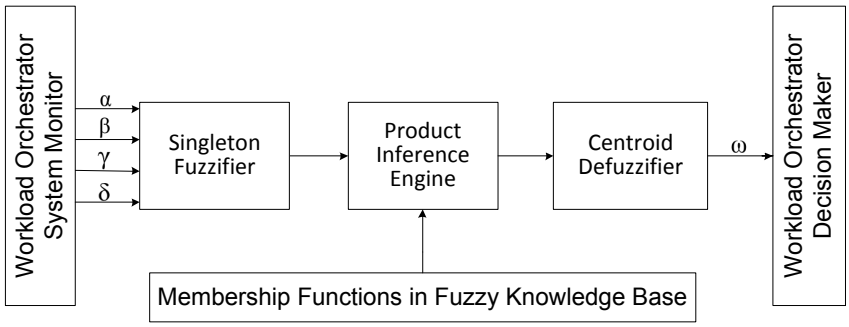


Figure 5.5. Fuzzy inference system for the proposed workload orchestrator.

real-world to scalar data [90]. We use a fuzzy logic-based orchestrator to map different metrics into a single value. The components of the proposed FLSs are singleton fuzzifier, product inference engine, and the centroid defuzzifier. These components are depicted in Figure 5.5. Essentially, both FLS used in both stages follow the same steps with different values. Since a more complex problem is solved in the second stage, the detailed explanation of the proposed FLS used in this stage is explained first.

5.3.2. Fuzzy Input Variables of the Second Stage

The second stage of our fuzzy logic based workload orchestrator operates on four input variables as shown below:

$$F = (\alpha, \beta, \gamma, \delta) \tag{5.1}$$

where α is *WAN Bandwidth*, β is *the length of the incoming tasks*, γ is the VM utilization on the edge server and δ is the delay sensitivity of related task.

Normally, any number of input variables can be used in fuzzy logic. In our work, we chose variables that have a significant effect on overall system performance. WAN bandwidth and the utilization of the edge servers are the essential factors that contribute to the probable bottleneck experienced in the system. Compared to these

factors, the probability that the WLAN itself or the computational resources on the cloud servers being a performance barrier is very low. Characteristics of the requested tasks of the applications also play a decisive role in the system performance. To provide a formal input representing the task properties in our fuzzy system, we make use of the length of the task and its sensitivity to delay, both of which are determined according to the profile of application creating the demand.

5.3.2.1. WAN Bandwidth. The WAN Bandwidth is an important indicator to considered while offloading tasks to the cloud server. For example, if the WAN communication latency is higher than the application QoS requirement or if the network congestion is too high to cause packet losses, offloading to the cloud server is not beneficial.

5.3.2.2. Task Length. The Length of Tasks essentially determines the execution time of the task. In our study, the length of a task is denoted by the number of instructions. It is assumed that the system has a load characterization approach to estimate the service type of the incoming tasks. The computational capacity of the VMs used in our simulations is measured in terms of GIPS, and the execution time of a task is calculated in accordance with its length. For example, if the length of a task is 100 million instructions (MI), it takes 0.01 seconds to execute the related task on the single-core server with 10 GIPS computational capacity. The task length can be used as a metric for the offloading decision. A complicated task containing too many instructions might be the right candidate for offloading to a cloud server. On the other hand, a simple task can be executed on the edge server or even on the end device itself.

5.3.2.3. Delay Sensitivity of Task. The delay sensitivity indicates the tolerance of the task to take a longer time to execute either due to network conditions or due to server utilization levels. Using this indicator while making "where to offload" decision is beneficial for increasing the overall QoS provided by the edge computing system. In our model, the delay sensitivity parameter is assumed to be determined by the application profile. In real-life examples, it can either be defined by the network administrator or

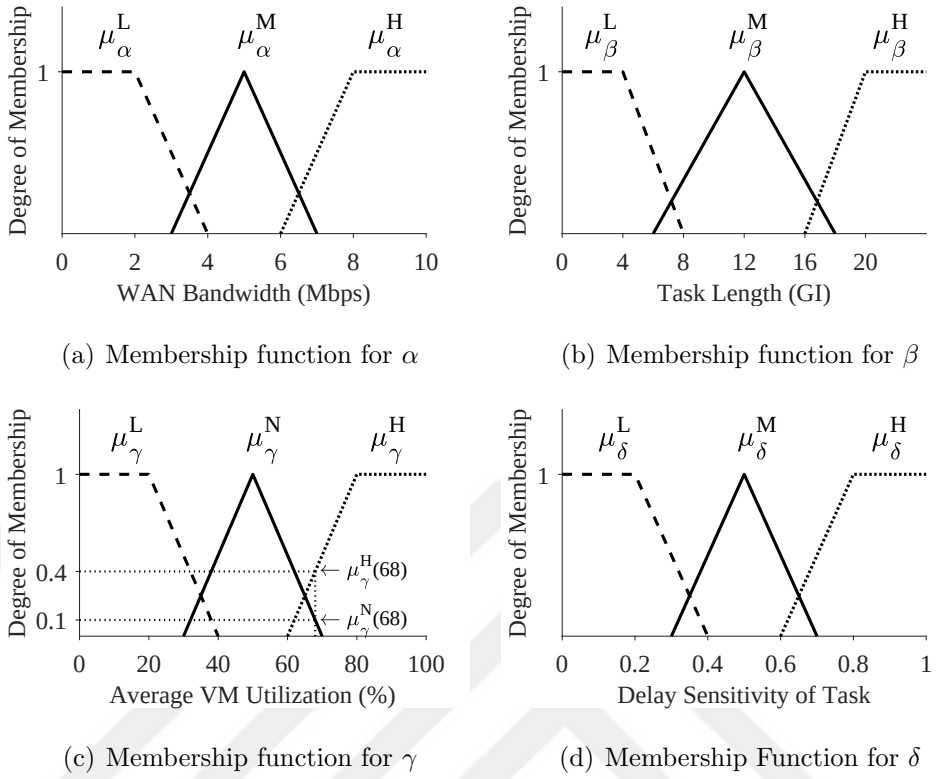
embedded into the tasks by the application developer.

5.3.2.4. VM Utilization. The VM utilization figure gives us information on the residual computational capacity of the edge server. The edge server can be considered as congested if the utilization is above a threshold level. In such a case, offloading to the central cloud server might be a feasible alternative depending on the WAN conditions and the delay sensitivity of the task. The edge servers run multiple VMs in our study. Therefore the VM utilization represents the average utilization of all VMs running on the related edge server.

5.3.3. Fuzzy Inference System

Inputs and outputs of the FLS are non-numeric linguistic variables. Instead of using the numerical values, FLS uses variables from the natural language. Our FLS has different linguistic variables for each indicator. For α, β and δ, we use *low* (L), *medium* (M), and *high* (H) as the linguistic variables. For γ, we use different terminology; here, we have *light* (L), *normal* (N), and *heavy* (H) as the linguistic variables.

The membership functions are defined to be used in fuzzification and defuzzification steps. We use the membership functions to quantify a linguistic term. For each fuzzy variable, a set of membership functions are defined, and this set consists of different functions for each linguistic term. In our case, we have four membership function sets, and each of the sets has a different function for related linguistic variables. The membership functions can be in different forms, such as triangular, trapezoidal, piecewise linear, Gaussian, or singleton [90]. We chose the triangular form, which is the most commonly used one. Our membership functions are depicted in Figure 5.6. Deciding the values used in the membership functions is a critical task; since it has a significant impact on the FLS performance. Similar to the other fuzzy decision mechanisms in other applications in the literature, the degree of membership values for each fuzzy variable is determined empirically. We tested various membership functions in our experiments, and the best combination among the trials is used.



(a) Membership function for α

(b) Membership function for β

(c) Membership function for γ

(d) Membership Function for δ

Figure 5.6. Membership functions for the fuzzy variables representing both network and computational resources.

The first step of FLS is the fuzzification step. The fuzzification is the process of transforming crisp values into fuzzy values by using the membership functions which associate a grade to each linguistic term. The fuzzifier defines fuzzy classes for each fuzzy variable as given below:

$$F_\alpha(x) = [\mu_\alpha^L(x), \mu_\alpha^M(x), \mu_\alpha^H(x)] \tag{5.2}$$

$$F_\beta(x) = [\mu_\beta^L(x), \mu_\beta^M(x), \mu_\beta^H(x) \tag{5.3}$$

$$F_\gamma(x) = [\mu_\gamma^L(x), \mu_\gamma^N(x), \mu_\alpha^H(x)] \tag{5.4}$$

$$F_\delta(x) = [\mu_\delta^L(x), \mu_\delta^M(x), \mu_\delta^H(x)] \tag{5.5}$$

Fuzzy values are calculated via membership functions of the fuzzifier, which are presented in Figure 5.6. For example, if the VM utilization on edge server is 68%, the degree of membership values of light, normal and heavy fuzzy variables will be 0, 0.1,

and 0.4 respectively, since $F_{\gamma}(68) = [0\ 0.1\ 0.4]$ as shown in Figure 5.6(c).

The second and the most crucial step of our FLS is the inference step. The inference is used to evaluate and combine the fuzzy rules from the fuzzy rule base. The result of the inference system is a fuzzy variable that is used in the defuzification step. A fuzzy rule is a simple IF-THEN rule with a condition and a conclusion. In fuzzy rules linguistic variables are used, such as *IF WAN Bandwidth is high AND Task Length is very high THEN offload to cloud*. Determining the fuzzy rules is very critical since the overall system performance highly depends on these rules. In our study, the relatively better fuzzy rule set is found empirically, and the best rule combination in the computational experiments are used. Since there are four membership functions with three linguistic terms, the number of fuzzy rules is 81. Some example rules are given in Table 5.1.

The aggregation, activation, and the accumulation methods are used in inference steps. The aggregation method, which is called as the combination as well, is the rule connection method. In aggregation phase, we use *Minimum* and *Maximum* functions for *AND* and *OR* operators respectively. The activation method defines how the evaluated IF the part result is applied to the THEN part of the rule. The most common activation functions are *Minimum* and *Product*. In the activation phase, we use the Minimum function. Finally, the accumulation method determines how the results of multiple rules are combined within a ruleset. Generally, the *Minimum* and *Maximum*

Table 5.1. Example fuzzy rules in the knowledge base.

Rule Index	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\gamma}$	$\boldsymbol{\delta}$	c
R1	low	low	light	high	edge
R2	low	high	normal	high	edge
R3	high	high	normal	low	cloud
R4	medium	medium	heavy	high	cloud
R5	high	high	heavy	high	cloud

functions are used as the accumulation method. In the accumulation phase, we use the Maximum function. The inference can be explained with a simple example. Assume that there are only five rules, $R1$ to $R5$, defined in our ruleset. If the WAN bandwidth, task length, VM utilization, and the delay sensitivity of the task values are given as 2 Mbps, 6 Giga instructions (GI), 35 percent, and 0.7 respectively, fuzzy value for selecting the edge server can be calculated by using Equation 5.6.

$$\mu_{edge} = max\left\{\mu_{edge}^{R1}, ..., \mu_{edge}^{Rn}\right\} \tag{5.6}$$

Since selecting the edge server is being analyzed in this example, the aggregation and activation phases are applied for $R1$ and $R2$:

$$\mu_{edge}^{R1} = min[\mu_{\alpha}^{L}(2), \mu_{\beta}^{L}(6), \mu_{\gamma}^{L}(35), \mu_{\delta}^{H}(0.7)] \tag{5.7}$$

$$\mu_{edge}^{R2} = min[\mu_{\alpha}^{L}(2), \mu_{\beta}^{H}(6), \mu_{\gamma}^{N}(35), \mu_{\delta}^{H}(0.7)] \tag{5.8}$$

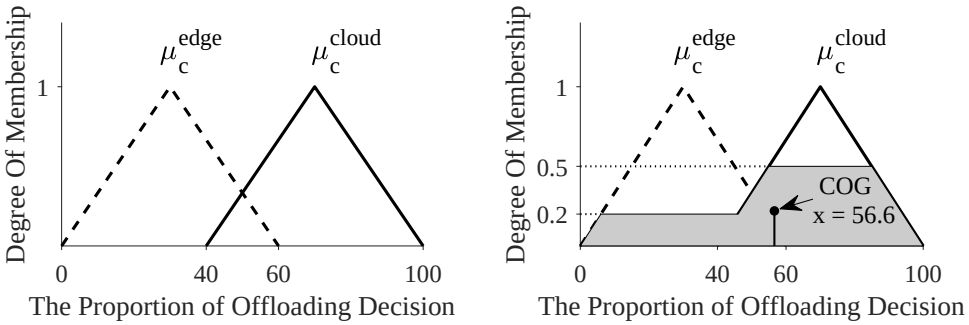
After calculating μ_{edge}^{R1} and μ_{edge}^{R2}, the fuzzy value for selecting the edge server can be found as 0.25 in the accumulation phase:

$$\mu_{edge}^{R1} = min[1, 0.5, 0.25, 0.5] = 0.25 \tag{5.9}$$

$$\mu_{edge}^{R2} = min[1, 0, 0.25, 0.5] = 0 \tag{5.10}$$

$$\mu_{edge} = max[0.25, 0] = 0.25 \tag{5.11}$$

The last step of the proposed FLS is the defuzzification step. Defuzzification is a process of transforming the fuzzy output inferred from the inference engine to a crisp value. We use a centroid defuzzifier in this step. The centroid defuzzifier returns the center of the inferred fuzzy outputs. The centroid calculation is the most popular defuzzification method, which returns the center of gravity of the area under the curve [91]. In Figure 5.7(a), membership functions of the consequence classes are depicted. The output of



(a) Output membership function (b) Center of gravity calculation

Figure 5.7. Membership function of the consequence classes.

the centroid defuzzifier can be calculated by using Equation 5.12.

$$\omega = \frac{\int x\mu(x)dx}{\int \mu(x)dx} \tag{5.12}$$

This method is similar to the formula for calculating the center of gravity. After applying the centroid defuzzifier, the output of the FLS, ω, becomes a crisp value between 0 and 100. We offload to the edge server if ω is less than 50. Otherwise, we offload to the cloud server. For example, if the μ_{edge} and μ_{cloud} are calculated as 0.2 and 0.5 by the fuzzy inference block, the crisp result will be 56.6, as shown in Figure 5.7(b). So, the incoming task is offloaded to the cloud.

5.3.4. Fuzzy Input Variables of the First Stage

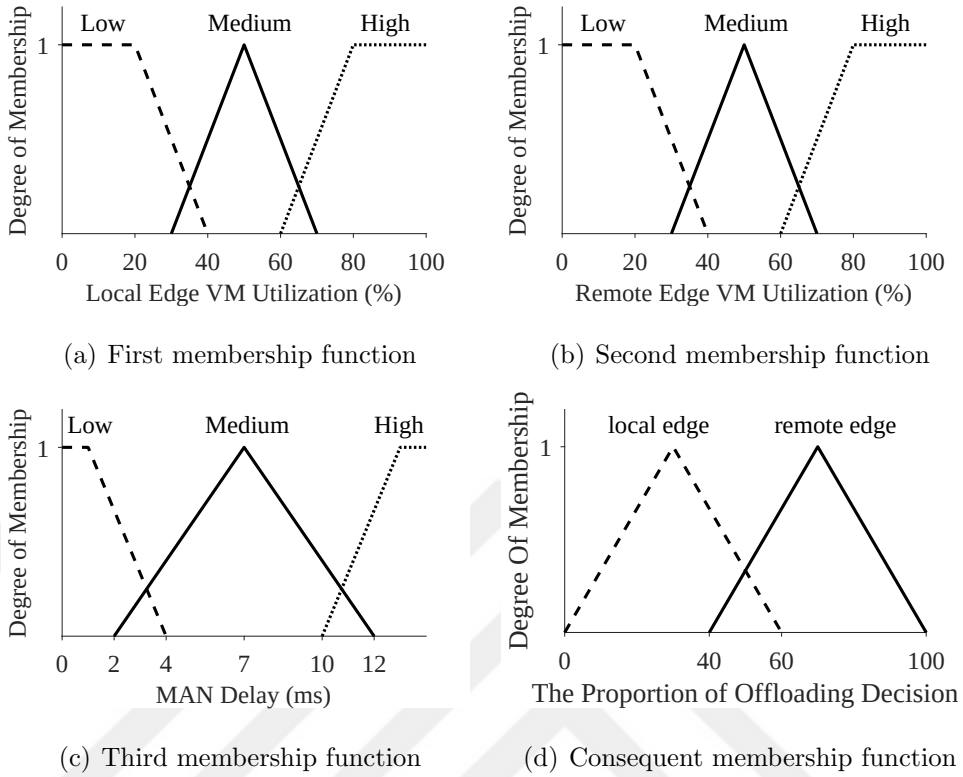
The fuzzy inference system used in the first stage is similar to the second stage. There are only three differences as the input variables, fuzzy membership functions, and the fuzzy rules. The number of fuzzy rules used in the first stage is 27 because there are three input variables with three linguistic variables each. The fuzzy rules are not given in this thesis, but the input variables and fuzzy membership functions are briefly addressed. In this stage, we use three input variables, which are MAN delay, CPU utilization of the least loaded remote edge server, and the CPU utilization of the local edge server.

5.3.4.1. MAN Delay. When the number of clients utilizing the MAN is very high, the MAN delay becomes an important bottleneck in the system. Therefore, we consider the MAN delay while deciding the most favorable edge server. If the MAN delay is higher than expected, it can be considered that the MAN resource is congested. In this case, offloading to the local edge server is advantageous.

5.3.4.2. Least Loaded Remote Edge CPU Utilization. The CPU utilization of the edge servers provides the status of computational resources. The MAN utilization cost of offloading to any remote edge server is the same. Therefore, we use the CPU utilization of the least loaded remote edge server. If the CPU utilization of the least loaded remote edge server is low compared to the local edge server, offloading to the remote edge server is advantageous if the MAN delay is low.

5.3.4.3. Local Edge CPU Utilization. Generally, offloading to the local edge server is beneficial in order not to consume the MAN resource. However, choosing the local edge is not rational if the number of users at different locations are not uniformly distributed. Always offloading to the local edge may cause too many task failures due to the inadequate computational resources, especially in the hotspot locations. If the MAN resource is not congested, distributing the incoming tasks among the edge servers may increase the system performance.

In the first stage of the proposed FLS, we use *low*, *medium*, and *high* as the linguistic variables. As it is explained, the corresponding membership functions are used for each fuzzy input variable. The membership function of the consequence classes is also used in the defuzzification step. The membership functions of the fuzzy input variables and consequence classes are shown in Figure 5.8.



(a) First membership function (b) Second membership function

(c) Third membership function (d) Consequent membership function

Figure 5.8. Membership functions used in the first stage.

5.3.5. Benchmark Solutions

The workload orchestration on the multi-tier edge computing in which the edge and cloud servers are used is a new concept. Therefore, a competitor workload orchestrator algorithm could not be found in the literature. Flores et al.'s work is one of the fuzzy logic-based approaches for code offloading [57]. Initially, their work utilizes the fuzzy logic to decide to execute the tasks on the mobile device or the cloud server. Whereas, our work assumes that the tasks are already offloaded from the mobile devices, and the fuzzy logic is used to decide for edge or cloud offloading. We adapted Flores et al.'s work in a way to execute their fuzzy logic on the edge server to decide on executing the tasks on the edge or cloud server. They use the *bandwidth*, *transferred data size*, *CPU speed* and *video execution ratio* as the input variables of FLS. All these variables are already available in our simulation environment except the video execution ratio, which indicates how much face recognition request is generated by the offloaded task. According to the explanation of authors, face recognition is a time-

critical operation that requires significant computational resources. It can be inferred that the video execution ratio determines the urgency type of compute-intensive tasks to be offloaded. In our simulation environment, we use the delay sensitivity variable to define the urgency of requests, and the augmented reality application has the highest delay sensitivity value. Therefore, the video execution ratio is a metric similar to the delay sensitivity of tasks. As a result, while adapting Flores et al.'s approach, we use the delay sensitivity value instead of the video execution ratio.

In addition to Flores et al.'s work, other competitor algorithms are also implemented. Generally, the WAN bandwidth and the CPU utilization of edge servers are the main bottlenecks of the multi-tier edge computing architectures. Therefore, they have a significant impact on overall system performance. We implement competitor algorithms that handle workload orchestration depending on the status of these parameters. This strategy is commonly used because it is easy to implement, and it does not bring a considerable overhead on the system. For example, load balancing in enterprise cloud services is done by considering the CPU utilization value. Our algorithms make use of *WAN Bandwidth*, *CPU utilization of VMs*, and *Hybrid (both bandwidth and CPU utilization)*. Bandwidth based method prefers to offload to the cloud servers if the WAN is not congested. Therefore the main aim of this algorithm is selecting the cloud server as long as the Internet connection is satisfactory. On the other hand, the CPU utilization based method favors offloading to the edge servers if the CPU utilization of VMs is not too high. The objective of this algorithm is selecting the edge servers as long as they are not congested in terms of CPU utilization. The Hybrid method is more complex when compared to others. It checks both the WAN bandwidth and the CPU utilization of VMs in the decision process. These algorithms use the threshold values for deciding to offload an incoming task to the edge or cloud server. The selected WAN bandwidth threshold should be as low as possible in order to maximize cloud offloading, but high enough to avoid network congestion. CPU utilization based method prevents offloading the incoming tasks to the edge server if the CPU utilization is higher than a specific threshold value. However, there is no utilization limit in bandwidth-based and hybrid methods, so they allow for full CPU utilization

on edge server-side. The threshold of average VM CPU utilization on the edge server should be as high as possible in order to maximize edge offloading, but low enough to prevent CPU congestion. We run various simulations with different parameters to find the minimum WAN bandwidth (*min-bw*) and the maximum CPU utilization (*max-cpu*) values, which provide the best performance. The candidate threshold values are 12, 8, 5, 3 Mbps and 90, 80, 70, 60 percent for *min-bw* and *max-cpu* in our experimental study. According to the simulation results, the best values for *min-bw* and *max-cpu* are observed as 5 Mbps and 80% respectivelly.

As explained in Section 5.3, the proposed workload orchestrator can offload an incoming task to the cloud, local edge server, or remote edge server. On the other hand, the competitor algorithms are only capable of determining the layer (edge or cloud) to be selected for task offloading. If the task is decided to be offloaded to the edge layer, the target edge server is selected via the least loaded algorithm. This algorithm checks the CPU utilization of all VMs on the edge layer and selects the VM with the least CPU usage. If MAN communication is not creating a significant delay, this approach can give successful results. Since we consider limited MAN resources in our decision logic, choosing a VM among all edge servers can cause a severe congestion problem on the MAN. We also evaluated different task provisioning algorithms on edge, such as first-fit, next-fit, and best-fit. The least loaded and the next-fit algorithms provide the best performance. We selected the least loaded algorithm for our system. If the objective function is different, such as minimizing the energy consumption on the edge servers, other sophisticated strategies can also be operated to find the optimum edge server to offload. The competitor workload orchestrator algorithms can be unified in a single algorithm, as given in Figure 5.9.

5.4. Performance Evaluation

We performed an extensive number of experiments to evaluate the performance of our fuzzy logic-based workload orchestrator. EdgeCloudSim [73], which can model a realistic edge computing environment, is used as the simulator. It provides a modular

Input: offload strategy S

Output: target layer to offload O

initialization:

read WAN bandwidth B

read average VM Utilization U

$O = edge$

if $S = BandwidthBased$ **then**

  if $B > B_{min}$ **then**

    $O = cloud$

  end if

else if $S = UtilizationBased$ **then**

  if $U > U_{max}$ **then**

    $O = cloud$

  end if

else if $S = Hybrid$ **then**

  if $B > B_{min}$ AND $U > U_{max}$ **then**

    $O = cloud$

  end if

else if $S = Flores*$ **then**

  $O = use\ modified\ Flores\ algorithm$

end if

return O

Figure 5.9. Unified algorithm representing competitor orchestrators.

architecture to provide support for a variety of crucial functions such as network modeling specific to WLAN and WAN, device mobility model, and realistic load generator.

5.4.1. Modeling Networking Delay

We use the result of an empirical study for the WLAN and WAN delays instead of a mathematical model to achieve a more realistic simulation environment by using

values from real life. However, the MAN delay is observed via a single server queue model with Markov-modulated Poisson process (MMPP) arrivals. The mean arrival rates of the tasks are updated when the state of the system congestion level is changed. It is assumed that all clients use a shared MAN which has Gigabit Ethernet like capacity. The clients use the MAN if they must be served by a remote edge server due to the congestion at their location. In this case, the edge server is accessed via two hops where the packets go through WLAN and MAN.

The most challenging part of this empirical study is dealing with a large number of real devices. To overcome this problem, we compared two different methods (i) taking measurements on five mobile devices (ii) use one computer which runs five parallel processes instead of using the real devices. It is observed that the results in both methods are very close to each other. Therefore, the experiments are performed by running parallel processes on one computer.

While measuring the WLAN bandwidth, two computers representing the server and client are used. The computer used as a server is connected to the access point via Gigabit Ethernet. The other computer representing the clients is connected to the access point via an 802.11 family WLAN. The available bandwidth value is calculated, while multiple clients (processes) are communicating with a server simultaneously. The average bandwidth of ten consecutive experiments is used for the WLAN bandwidth.

While measuring the WAN bandwidth, an empirical study is carried out for characterizing the typical home/office Internet connection capacity in Istanbul, Turkey. In this study, the static files located in Amazon simple storage service (S3) are downloaded simultaneously by the multiple clients. The WAN bandwidth values are determined by the average values of the measurements taken at different times of the day for a week. The results of the WLAN and WAN bandwidth measurements are shown in Figure 5.10. The number of clients given in Figure 5.10 is not the number of mobile devices connected to the same access point. In fact, that number indicates the number of mobile devices communicating with the edge or cloud server at the same time.

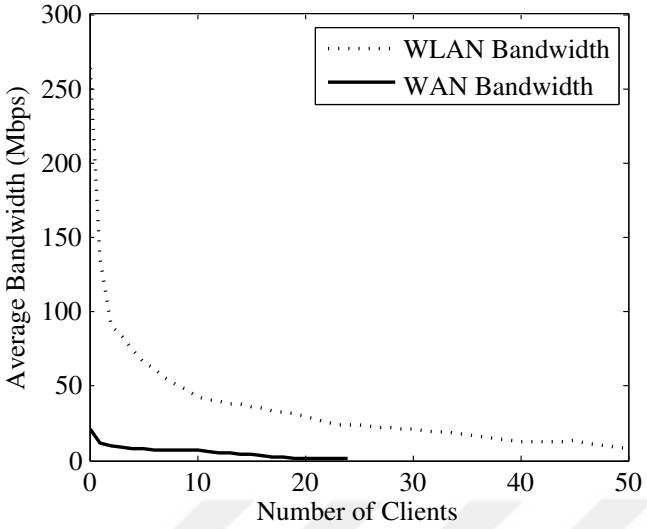


Figure 5.10. Empirical evaluation of WAN/WLAN connections.

We observed that it is unlikely to transmit data when more than a certain number of clients use the same network resources simultaneously. Therefore, it is assumed that the packets are dropped due to the congestion when the maximum number of clients that can use the network resources is reached. This number is determined as 25 for the WAN and 50 for the WLAN communication.

5.4.2. Simulation Setup

In a generic edge computing environment, mobile devices can run several services/applications. The characteristics of the potential applications may vary according to the requirements. The task offloading request to the edge or cloud server generates a different level of CPU and network load on the system. For instance, a heavy computational task requires a large number of CPU resources, whereas a data backup application requires a small number of CPU resources. On the other hand, the data backup process needs a significant amount of data transmission, while the compute-intensive task may not use too much network capacity. In order to simulate real-life, more realistically, we use four different application types in our study. To decide on the application types, we considered the most presented edge computing use cases given in

the literature. For example, Silva et al. propose an augmented reality application on Google Glass [75], Guo et al. discuss the infotainment applications [76], and Tunca et al. present a health application which uses a foot-mounted inertial sensor to analyze the walking pattern of the users [77]. In our simulations, it is assumed that the mobile devices offload tasks that belong to a predefined set of application categories. The user wearing the smart glass offloads captured pictures to the remote servers, which provide the face recognition service. The users who are carrying a foot-mounted inertial sensor offload the sensor data to the remote servers, which provide a fall risk detection service. Similarly, infotainment and the compute-intensive applications send their tasks to the remote servers which provide related services.

The characteristics of application types used in the simulations are listed in Table 5.2. The *usage percentage* of the application stands for the portion of the mobile devices running this application. The *task interarrival* time defines how frequently the related task is sent to the edge orchestrator. In our study, the interarrival time of the tasks is exponentially distributed. The *delay sensitivity* is basically used for determining if the task is delay intolerant (real-time) or not. If this value is high, the related application is delay intolerant. The *active/idle* period is used while generating the

Table 5.2. Application types used in the simulations.

	Augmented Reality	**Health App**	**Compute Intensive**	**Infotain. App**
Usage Percentage (%)	30	20	20	30
Task Interarrival (sec)	2	3	20	7
Delay Sensitivity (%)	0.9	0.7	0.1	0.3
Active/Idle Period (sec)	40/20	45/90	60/120	30/45
Upload/Download Data (KB)	1500/25	20/1250	2500/200	25/1000
Task Length (GI)	9	3	45	15
VM Utilization on Edge (%)	6	2	30	10
VM Utilization on Cloud (%)	0.6	0.2	3	1

Table 5.3. Simulation parameters.

Parameter	**Value**
Simulation Time/Warm-up Period	33/3 minutes
Number of repetitions	60
WAN/WLAN Bandwidth	empirical
MAN Bandwidth	MMPP/M/1 model
LAN Propagation Delay	5 milliseconds
Number of VMs per Edge/Cloud server	8/4
Number of Cores per Edge/Cloud VM CPU	2/4
VM CPU Speed per Edge/Cloud	10/100 GIPS
Mobility Model	Random Way Point
Probability of selecting a location type	Equal
Number of locations Type1/2/3	2/4/8
Mean dwell time in Type 1/2/3	2/5/8 minutes

task. Related mobile device generates tasks during the *active* period, and it just waits in the *idle* period. The *upload/download* data sizes depict the typical values for data sent to/received from the server. For instance, 1500/25 KB upload/download size is chosen for augmented reality to reflect the fact that an image is uploaded, and text metadata is received as the response. The *task length* determines the required CPU resource for the related task in the GI unit. Since the computational capacity of the server is measured in terms of its GIPS value, the task size is a determinant of the duration of its execution. Like the task interarrival time, the length of the tasks is also an exponentially distributed random variable. The *VM utilization* parameter is the CPU overhead on the VM while the related tasks are running on it. The VMs are capacitated in terms of CPU in our simulations. If there is not enough CPU resource on the VM for the incoming task, it will fail.

Other important simulation parameters are listed in Table 5.3. We assume that residents move within the campus according to a nomadic mobility model [78]. The

nomadic model dictates a user goes to a location stay there and moves to another location. In our model, there are three location types with different attractiveness levels, and the residents stay in these locations for random dwell times proportional to the attractiveness level. Average dwell values for each location is given in Table 5.3.

In real-life scenarios, the applications depict an intermittent behavior where they actively generate tasks for a duration and stay idle between active periods. For example, the health application may record the sensor data for a while, and then it sends the collected data to the remote server for further processing. We model this pattern using values presented in Table 5.2. It is assumed that each location is covered by a dedicated wireless access point, and the mobile devices join related WLAN when they move to the related location. After joining the WLAN, mobile devices start sending tasks to the edge or cloud server. If the task is offloaded to the cloud server, the broadband connection provided by the wireless access point is used. Whereas, MAN connection is used if the task is offloaded to the remote edge server. In this scenario, MAN can be a Gigabit Ethernet campus LAN.

In this thesis, the orchestrator is used in the edge of the network, which manages the tasks sent to it. In our previous work, we investigated other architectures where the edge orchestrator is not available [92]. When there is no edge orchestrator, all the tasks are executed at the local edge server, which is connected to the same access point with the mobile device. Therefore, the resources are not utilized efficiently, and high congestion is observed on edge servers located at the hotspot locations. The result of our previous work is not presented in this study, because its performance is too poor to compare with the proposed algorithms.

5.4.3. Simulation Results and Discussions

The results of our simulation in EdgeCloudSim consist of the average number of failed tasks, the average service time, the average VM utilization, and the average network delay. In our figures, we use *Flores**, *fuzzy*, *util.*, *bw*, and *hybrid* abbreviations for

Modified Flores-Based, *Fuzzy-Based*, *Utilization-Based*, *Bandwidth-Based*, and *Hybrid* workload orchestrators respectively. The main performance criteria are compared in Figure 5.11 to Figure 5.13, considering the average results of all applications.

5.4.3.1. Failed Task Performance. The average unsuccessful packet or task ratio is one of the most important performance criteria in computer network design. The average failed task percentage in our architecture is given in Figure 5.11. The performance of all approaches is similar when the system is lightly loaded. However, the *Fuzzy-Based* approach provides better results when compared to its competitors as the load increases. It can balance both computational and communication (networking) resources better than its competitors. When it comes to the other approaches, one of the main reasons for providing poor performance is MAN congestion. Since the competitors do not consider the status of the MAN resources, they experience too many network losses on the MAN when the system is loaded. The *Utilization-Based* approach provides the worst results because it uses CPU utilization as a threshold, and after the computational resource of VMs is congested, incoming tasks are sent to the cloud. As a result, after some time, most of the tasks drop on the network due to the WAN congestion.

5.4.3.2. Service Time Performance. Another important performance metric is the service time since the completion time of an objective is generally significant for many users. The average service time of our algorithms is shown in Figure 5.12. The service time includes both the network delay and the processing time. When the system load is light, the *Fuzzy-Based* approach provides the best result because it considers both computational and network resources and makes better decisions. Since the *Bandwidth-Based* approach prefers offloading the tasks to the cloud, it provides the worst performance due to the WAN delay. When the system load is high, both the processing time of the tasks and network delay are getting higher due to the congestion. In this case, especially the WAN resources become congested. The *Bandwidth-Based* and *Modified Flores-Based* approach prefer offloading the tasks to the edge, so they provide better results. The *Utilization-Based* and *Hybrid* algorithms decide to offload

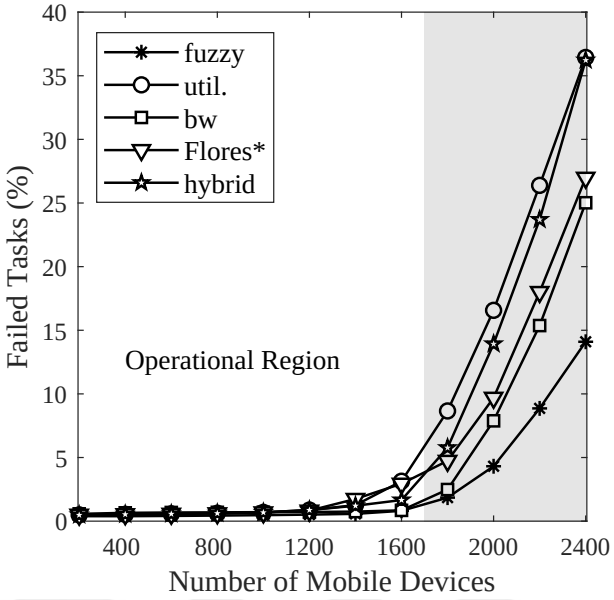


Figure 5.11. Average failed task based on all application types.

to the cloud due to the congestion on CPU resources on the edge servers. Since the WAN delay is higher than the processing time, they provide the worst performance. It can be argued that the average service time of the *Fuzzy-Based* approach becomes worse than the others when the number of mobile users is higher than 1800. However, in delay and loss systems, the average loss and delay should be considered together. In the shaded region of Figure 5.11, it can be seen that a considerable amount of tasks cannot be served by the competitor algorithms. Therefore, the *Fuzzy-Based* approach provides longer service time than the others if the system is overloaded, as shown in the shaded region of Figure 5.12. Considering the average number of failed tasks, it is quite likely that the desired operational range of this particular instance is limited to 1800 mobile devices. If the number of mobile devices is too high, the ceiling effect in the average service time results is more likely to be experienced by all algorithms, since most of the requests are blocked.

5.4.3.3. CPU Utilization Performance. The average CPU utilization of VMs running on the edge servers are shown in Figure 5.13. Providing low CPU utilization is not

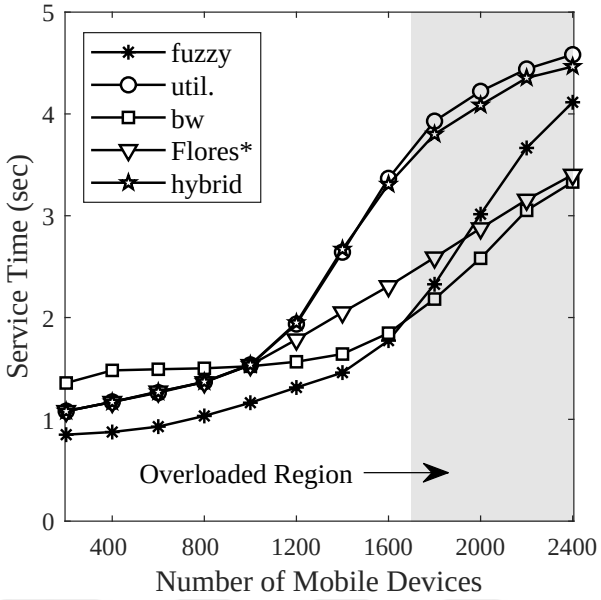


Figure 5.12. Average service time based on all application types.

always the desired result. If the system has lower CPU utilization with more VMs, it means that there is a waste of resources. However, if it has lower CPU utilization by using the same number of VMs as the competitors, it means that the related system uses resources more efficiently. According to the average CPU utilization results of our algorithms, it can be seen that the *Fuzzy-Based* and *Bandwidth-Based* approaches use computational resources of VMs better than the other algorithms when the system is not heavily loaded. For all the algorithms except the *Fuzzy-Based*, the average CPU utilization increases to a certain point, then it becomes stationary, and finally, a decreasing trend is observed. Since, if the system is heavily loaded, the average CPU utilization in hotspot locations becomes very high, but the edge servers located at the locations with low attractiveness levels could not be utilized well by the competitor algorithms due to the MAN communication failures. Since the *Fuzzy-Based* approach considers the MAN delay, it can adapt to the dynamic environment. As a result, it utilizes the edge servers more efficiently than its competitors.

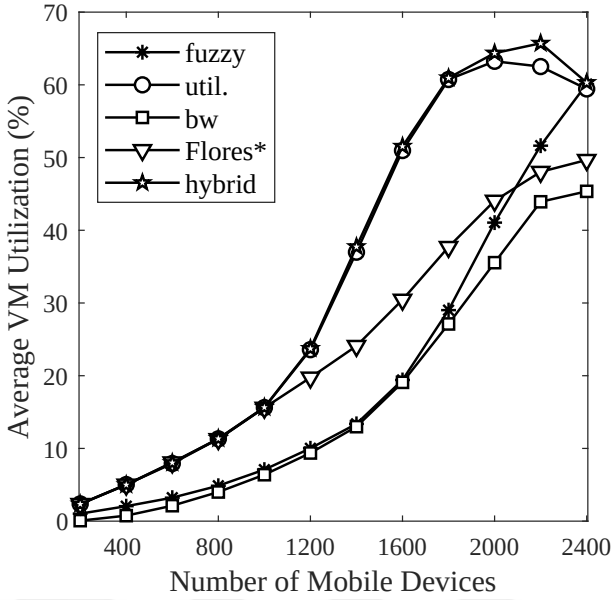


Figure 5.13. Average VM utilization based on all application types.

5.4.3.4. Task Failure Analysis. In our simulations, the tasks fail due to four reasons which are the mobility, lack of VM capacity on the edge servers, lack of MAN, and the WAN capacity. In theory, the lack of WLAN capacity also causes task failures, but very few task losses due to the WLAN capacity are observed for the cases studied. The proportion of task failures for 2000 mobile devices is shown in Figure 5.14. The failures due to the mobility are unavoidable since the task level handoff is not considered due to the real-time properties of the target applications. If the users change their location before getting the response of the previously requested tasks, related tasks fail because the users are not connected to the related WLAN anymore. So, for all the algorithms, we can observe a somewhat similar number of failed tasks due to the mobility on average. However, the ratio of this kind of failure may vary according to the other task loss results. First of all, MAN communication failure is a common problem for all competitor algorithms since they do not take the MAN capacity into consideration while selecting a server in the edge layer. In the *Utilization-Based* approach, the majority of the tasks drop because of the WAN congestion. Hence it only considers the CPU utilization of the edge VMs and starts offloading tasks to the cloud server after

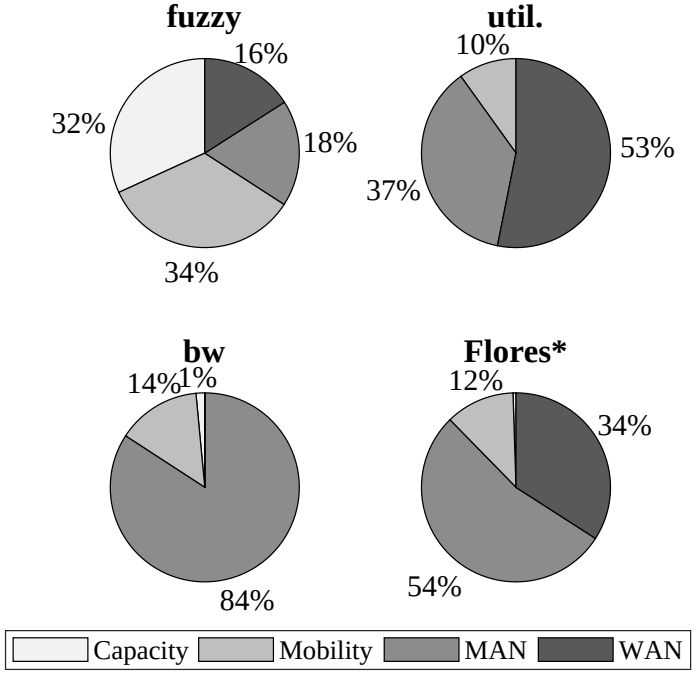
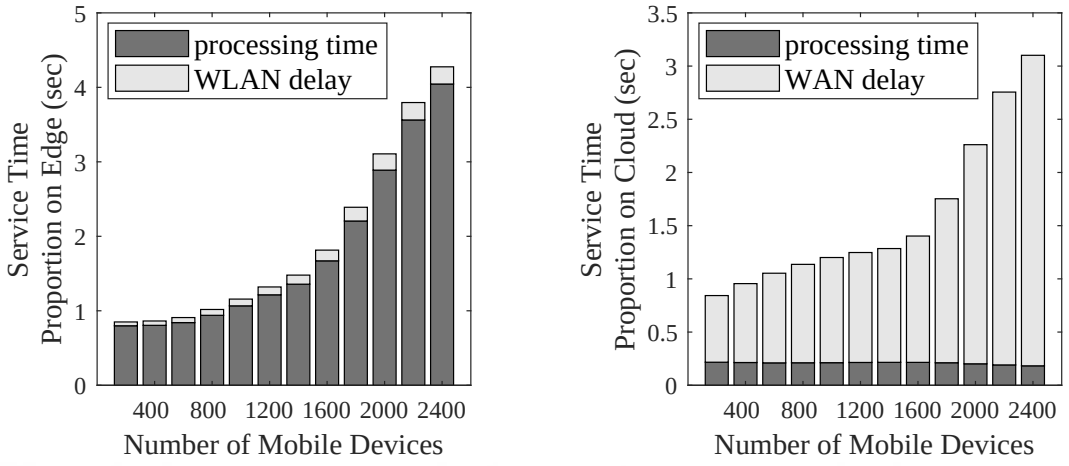


Figure 5.14. Distribution of task failure reasons.

the VMs are congested. On the other hand, in the *Bandwidth-Based* approach, most of the tasks drop due to the MAN capacity, because it only considers the WAN bandwidth and starts offloading tasks to the edge server after the WAN is congested. When it comes to the *Modified Flores-Based* approach, the impact of the MAN congestion on the task failures is more than that of the WAN congestion. Finally, in the *Fuzzy-Based* approach, the losses are more balanced than the other algorithms.

5.4.3.5. Service Time Analysis. The service time includes both processing delay and the communication delay. If the task is executed on the edge server, the total service time consists of the WLAN/MAN delay and the edge VM processing delay. On the other hand, if the task is offloaded to the cloud server, the total service time consists of the WAN delay and the cloud VM processing delay. In Figure 5.15, service time score of the tasks are provided by considering both the processing delay and the communication delay for the *Fuzzy-Based* algorithm. The results for the tasks which are executed on the edge server are shown in Figure 5.15(a). The dominant time period consumed on



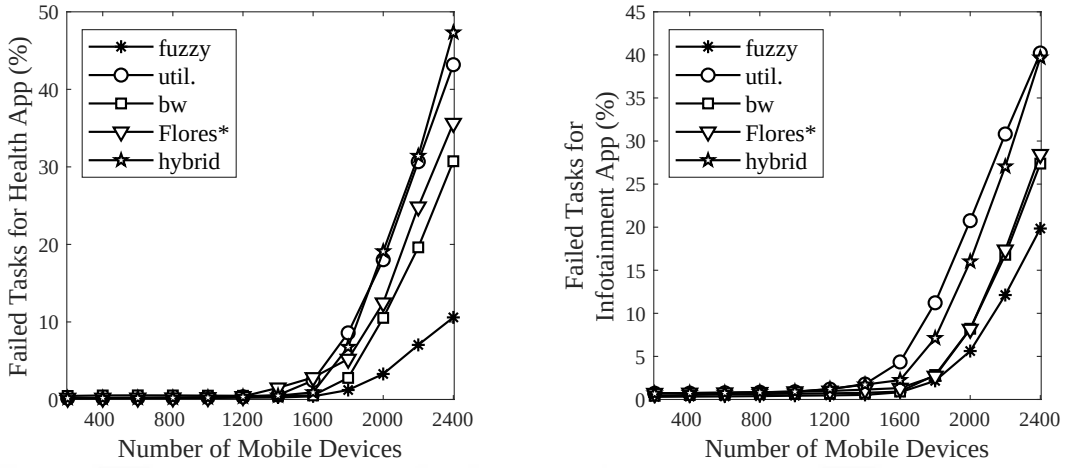
(a) Proportion of service time on edge

(b) Proportion of service time on cloud

Figure 5.15. Average service time values for fuzzy based workload orchestrator.

the edge server offloading process is the processing time. As expected, the WLAN communication time is very low. As the load increases, both the processing and the networking time rises, but the increase rate of the processing time is much more than the networking time. The results for the tasks which are executed on the cloud server are shown in Figure 5.15(b). In this case, the dominant time period spent on the cloud server offloading process is the WAN communication time. Since the VMs running on the cloud servers are very powerful in our scenario, the processing time is low. As the load increases, only the networking time increases. In fact, in the worst case, it may take 5 to 6 seconds for the *Utilization-Based* algorithm.

5.4.3.6. Application Performance Analysis. Instead of the average results for all application types, we can separately investigate the performance of the algorithms for each application. In Figure 5.16 and Figure 5.17, the result of different applications are compared. In Figure 5.16(a) and 5.16(b), the average failed task performance of the algorithms is evaluated by considering the health and the infotainment applications. As explained in Section 5.4.2, the infotainment application generates big tasks (results in 10% CPU utilization on the corresponding edge VM), whereas the health application generates small tasks (results in 2% CPU utilization on the corresponding edge VM). In addition, the tasks generated by the health application have a higher responsiveness

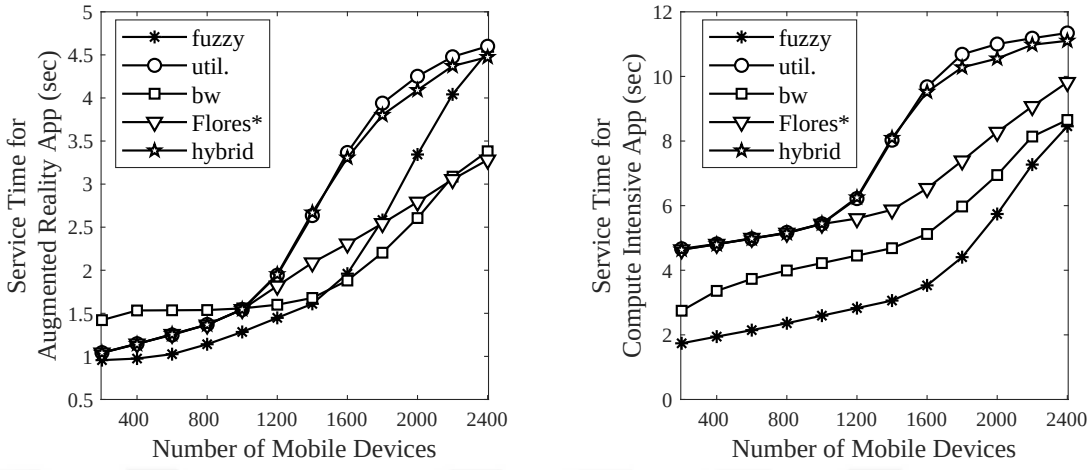


(a) Health app. score

(b) Infotainment app. score

Figure 5.16. Average failed task for different application types.

level. According to the results, the *Fuzzy-Based* algorithm outperforms others while serving time-critical small tasks. We can say that our fuzzy based algorithm distinguishes the tasks by their size and responsiveness level better than its competitors. It also provides better performance for the big tasks. In Figure 5.17(a) and 5.17(b), the average service time performance of the algorithms is evaluated by considering the augmented reality, and the compute-intensive applications. The augmented reality application is a delay intolerant application, and it generates medium-sized tasks (results in 6% CPU utilization on the corresponding edge VM). Whereas, the compute-intensive application is a delay tolerant application, and it generates very big tasks (results in 30% CPU utilization on the corresponding edge VM). Among the competitor algorithms, the *Bandwidth-Based* algorithm provides better service delay performance for the compute-intensive application. It can be said that executing the tasks generated by the compute-intensive application mostly on the cloud servers provides better performance. The *Fuzzy-Based* algorithm provides the lowest service time since it also uses the edge servers when they are lightly loaded. When it comes to the augmented reality application, the *Fuzzy-Based* approach provides better results when the number of mobile devices is low. If the system is very congested (e.g. 2400 devices are used), the *Fuzzy-Based* approach has the highest service time. The explanation for Figure 5.12 is also valid for this case. If the system is heavily loaded, the *Fuzzy-Based* solution



(a) Augmented reality app. score (b) Compute intensive app. score

Figure 5.17. Average service time for different application types.

can execute more augmented reality application tasks than the competitors. Since it does not loose service requests, it is expected that the *Fuzzy-Based* approach will have higher service response times when the system is congested. This situation is observed on the augmented reality application because the number of tasks generated by this application is higher compared to the compute-intensive application.

6. MACHINE LEARNING BASED WORKLOAD ORCHESTRATOR FOR VEHICULAR EDGE COMPUTING

We are witnessing the new era of connected vehicles where transportation is undergoing a radical transformation. Vehicles of varying purpose, functionality and dimension, whether it be an airport shuttle bus, a fleet of trucks or a personal sports car are being transformed into intelligent, autonomous entities that are equipped with many sensors, on-board computational units and versatile communication capabilities that enable them to cooperate with the road-side infrastructure and/or with each other [93]. It is predicted that, by 2030, 95% of all U.S. passenger miles will be served by transport-as-a-service (TaaS) providers who will own and operate fleets of autonomous electric vehicles providing passengers with higher levels of service [94].

Triggered by the autonomy, a new in-vehicle experience is defined for the passengers where the transportation time is planned as if individuals are in a stationary domestic environment or in an office space [95]. In this setting, passengers are expected to not concentrate on the actual travel but instead accomplish daily tasks in an accordingly designed interior space where infotainment and productivity applications will be of primary use. This novel vision, however, comes with a trade-off in which extensive amounts of data are produced on the vehicle itself that have to be processed with stringent real-time requirements. An autonomous vehicle fitted with imaging and scanning systems generates and processes about 4 TB of data for every hour of autonomous driving [96]. Moreover, crucial data related to the vehicular applications typically exist in a distributed fashion over the vehicles and the road-side infrastructure, which necessitates timely communication between all parties. Apart from the sensory processing, the number of in-vehicle applications used by the drivers as well as the vehicle itself has increased accordingly as the vehicles evolved in the recent years by the help of the new generation IoV services such as the intelligent navigation and cognitive driver assistance. With this spectrum of increasingly sophisticated services,

the on-board computational capabilities of the vehicles may become inadequate [97].

VEC is an emerging technology offering a basis for the in-vehicle applications to improve the availability of services using the computation offloading concept [98]. VEC improves the user experience by partially delegating the load via computational offloading to an edge infrastructure streamlined for vehicular applications. Thus, it can handle the delay intolerant complex operations. The advanced driver assistance [99], e-horizon [100], autonomous driving [101], and accident prevention applications [102] have the highest potential for the VEC based systems. These applications generally use ML, and AI-based algorithms that process real-time data and require a significant computation power [103]. The vehicles cannot execute such algorithms efficiently due to insufficient local computing resources.

The concept of IoV, its pioneering applications, and the services for the future smart highways can benefit from the computation offloading concept over a multi-tier architecture consisting of the connected vehicles, RSUs, and cloud computing elements as shown in Figure 6.1. The vehicles are located in the first tier, which can be considered as the data generation layer. They also have computational resources which are provided by their OBUs. If required, some of the operations can be executed locally by the OBUs at this tier. The second tier consists of the RSUs that can provide fast access to limited resources. The edge servers are located in this tier. Finally, the third tier includes traditional cloud computing elements.

While the concept of task offloading offers great potential by augmenting the computational capabilities of the vehicles, it is not straightforward to design an offloading based system. Firstly, a very dynamic scene is present where a multiplicity of novel applications with varying loads and requirements are to take advantage of the infrastructure. Secondly, an overall transient operational character is inevitable due to the short residence times of the vehicles under the coverage of fixed edge server components such as RSU. Moreover, the overall system involves the cooperation of local edge servers with the global cloud servers distributed over a geographical region. While this

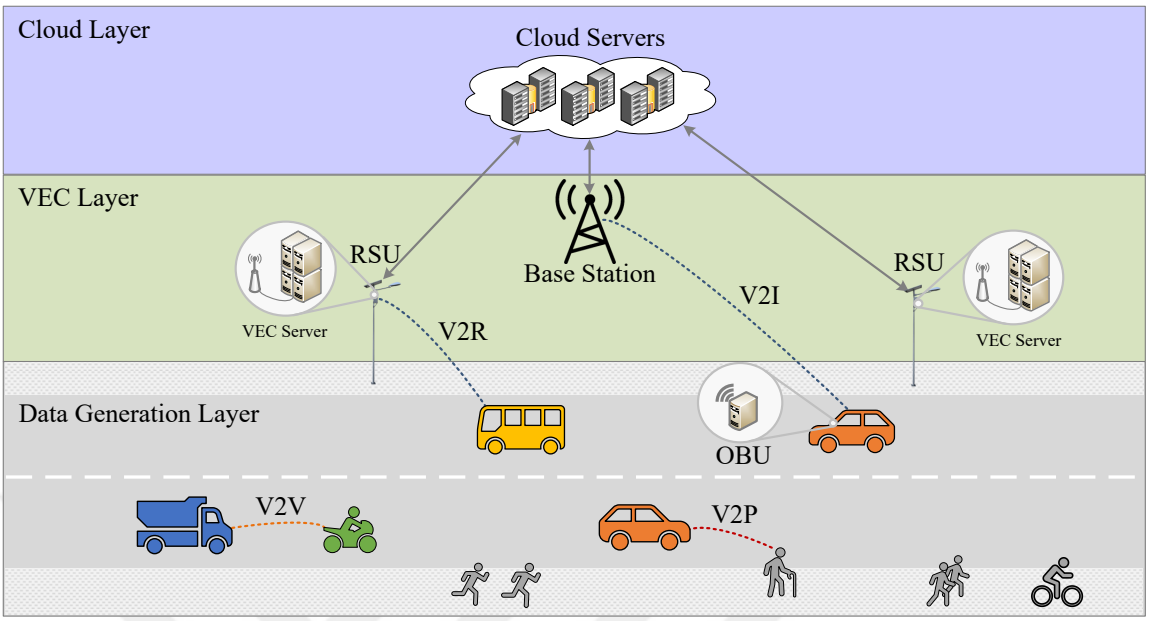


Figure 6.1. Multi-tier VEC architecture for vehicular networks.

multi-tier setting increases efficiency, it also increases the state-space, thus, contributes to the overall complexity of the instantaneous decisions that are to be given by the system. As a result, an efficient workload orchestration and task scheduling scheme become a crucial requirement.

The computational and network resources on the VEC systems are heterogeneous and change very quickly. As the number of vehicles increases, the congestion on these resources becomes inevitable, which makes the decision making process more difficult [104]. Therefore, predicting the status of resources by using historical data is a challenging process. The ML-based prediction is one of the efficient methods that can be utilized in such highly dynamic environments. Zhan et al. proposed a deep reinforcement learning-based offloading scheduling for VEC [105], and Liu et al. proposed a deep learning-based V2X wireless channel prediction model using a long short-term memory (LSTM) network [106]. While ML-based approaches have been applied to various problems in many areas, they have found only limited use for VEC systems. We focus on the workload orchestration problem in the VEC environments and different

ML algorithms to fill the gap in this field. To the best of our knowledge, we propose the first ML-based workload orchestrator for the multi-tier multi-access VEC architecture used in this work. The main contributions of this paper are summarized as follows:

- We define a multi-access multi-tier VEC architecture. The multi-access communication technologies consist of vehicular WLAN, WAN, and cellular network, which allow vehicles to apply V2I offloading. The multi-tier architecture has three tiers, including vehicles, RSUs, and cloud servers.
- We propose an ML-based workload orchestration approach for VEC systems. In particular, the ML-based orchestrator performs a two-stage process. In the first stage, a classifier model predicts whether the results of the offloading options are successful or not for each target device. In the second stage, a regression model estimates the service time of the related options. Finally, the target device which promises the lowest service time is selected.
- We experimented with multiple classifiers, including naive Bayes (NB), support vector machine (SVM) and MLP models. The MLP was chosen as the best classifier based on the performance comparison of these models.
- We carried out detailed simulation experiments to evaluate the performance of the proposed workload orchestrator. We consider load/congestion dependent communication and computing delays. The simulation results show that our approach outperforms its competitors regarding the task failure rate and service time.

The remainder of this chapter is structured as follows. Section 6.1 explains the proposed ML-based workload orchestrator for multi-access multi-tier VEC architectures. Section 6.2 presents the simulation experiment design and analyzes the results.

6.1. ML-Based Vehicular Workload Orchestrator

Since the service requests are created one after the other in a dynamic fashion, workload orchestration is an online problem and cannot be solved by the formal optimization tools like CPLEX and Gurobi. Instead, we propose an ML-based orchestrator

that tries to maximize the percentage of satisfied services by dynamically changing network conditions and server utilizations.

A VEC system is typically composed of a multitude of edge servers accompanied by cloud resources. Similarly, the communication infrastructure required for VEC incorporates WLAN, MAN, and WAN technologies. Heterogeneous infrastructure combined with intermittent and bursty task requests creates a substantially dynamic scene in terms of task processing. In this setting, the edge orchestrator tries to maintain an efficient operation of the overall system by deciding on the best available computational unit to offload. This decision is considerably challenging as it should take into account both computational and networking states as well as the workload characterization.

6.1.1. Multi-Tier VEC Architecture

The access technologies used in the VNs and the topology of infrastructure have a considerable effect on the system performance. We think that future smart highways will use the advanced applications on the multi-tier multi-access VEC architectures, which include both edge and cloud servers, as shown in Figure 6.2. A similar MEC-based vehicular network model is proposed in other studies as well. Peng et al. [107] and Sami et al. [108] use a cellular network for V2I communication and the IEEE 802.11 WLAN protocol for V2R communication. In our architecture, the vehicles can offload their tasks to the edge servers by using the WLAN interface. A task can also be offloaded to the cloud servers via the Internet connection (WAN), which offers the possibility of providing more flexible network infrastructure. In the proposed architecture, there are two alternatives for the broadband connection, one via RSU and the other via the cellular network. The RSUs can access to the cloud resources in a multi-hop manner by using the cellular base station as proposed in [109]. However, this approach creates a significant bottleneck on the base stations. In our proposal, the RSUs use a fiber communication link to provide the Internet connection. Offloading to edge, cloud via RSU, and cloud via cellular network (CN) scenarios are illustrated in Figure 6.2 as cases 1, 2, and 3 respectively.

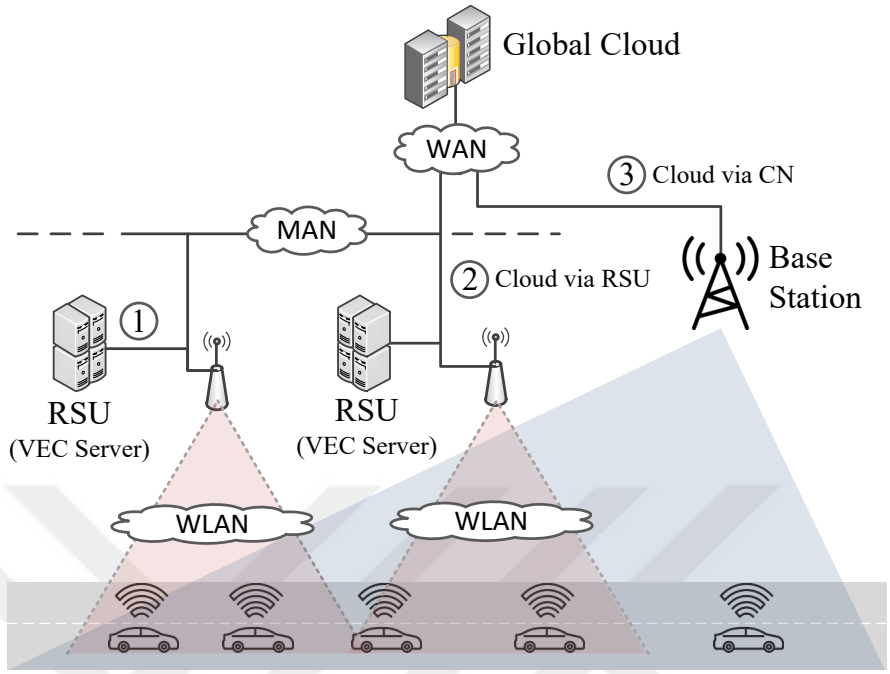


Figure 6.2. Multi-tier multi-access VEC architecture and task offloading options.

The RSUs are also connected to a MAN in the proposed architecture. The MAN connectivity allows RSUs to share their computational capacity via task migration. Therefore, the VEC layer can be thought of as a shared resource pool. In addition, MAN offers a solution to the handover problem. We use the same handover approach proposed in [63]. If the vehicle leaves the range of the serving RSU before the offloaded task is executed, the result is transmitted to the related vehicle in a multi-hop manner through the other RSUs. The handover operation only fails when the vehicle leaves the range of the serving RSU while transmitting data. Although the network delay on MAN is very small and commonly ignored, we take this delay into consideration.

6.1.2. Two-Stage ML-Based Vehicular Edge Orchestrator

As already stated, using the traditional approaches for the workload orchestration cannot provide the desired performance on the dynamic environments where the state of resources changes rapidly. Therefore, different techniques such as ML or AI-based solutions are recommended for the workload orchestration [59]. In our work, we propose

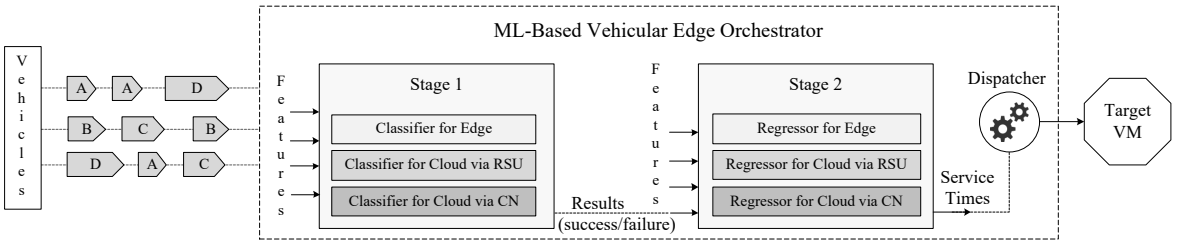


Figure 6.3. Two stage ML-based vehicular edge orchestrator.

an ML-based orchestrator that has two-stages to decide where to offload, as shown in Figure 6.3. In the first stage, three classification models are used for each option to estimate the result (success or failure) of the offloading process. In the second stage, the service time is estimated by using related regression models for the options predicted to be successful in the first stage. We performed a simulation experiment to collect the training data via EdgeCloudSim [73]. A random orchestration policy is applied under variable load by recording various features (input variables) that can be used in the classification and regression models. Then, the possible features were investigated, and the ones which have more impact on the result of algorithms were selected.

We opted to utilize a two-stage architecture because the success/failure and the service time of a task are related but different metrics. The service time is meaningful and defined only if a task is successfully executed. It is possible to train a single model that could infer both metrics by assigning large ground truth service time values for failed tasks; however, that would mean alteration of the ground truth, which could introduce artifacts in the data, and would produce an unnecessarily complex model with potentially worse generalizability. Dividing the problem into two sub-problems allows us to use two simpler models that are easier to tune and generalize. Another advantage is to be able to use different input features for the two problems, which may indeed be impacted by different criteria.

In essence, ML-based solutions have already been studied in various domains in recent years. Our two-stage ML-based vehicular edge orchestrator aims at increasing

the number of successfully offloaded tasks and reducing the service time as much as possible. The first stage predicts whether offloading to a specific location would result in failure. That location could be eliminated from the decision-making in the second stage, which predicts the service time. As a result, although the classification and regression models are already used in many areas, the novelty of our work is using these models successively for the vehicular edge orchestration problem.

6.1.3. Classification Models

The classification model is used to forecast whether the offloaded task to a particular processing unit will be successful or not. The task offloading can be considered as successful if the task execution and transmission (including both the upload and download cases) succeed. Considering the overall system performance of the cases studied, predicting the result of the offloading operation has more impact than predicting the service time. Therefore, the most crucial stage of the proposed ML-based edge orchestrator can be considered as the classification stage. We inspected three classification algorithms, which are NB, SVM and MLP. All the employed models are supervised, i.e., the ground truth class labels of the instances are used to train the models.

NB is a probabilistic classifier that is based on the Bayes theorem [110]. Each class is represented with a multivariate Gaussian probability with respect to the features, with the assumption of independence between the features. Hence the covariance of the fitted Gaussians is diagonal.

SVM is a binary (two-class) classifier that aims to determine a separating hyperplane between the classes that maximizes its margin to the class instances. Hence it uses a solver to determine this discriminating hyperplane. We specifically use John Platt's sequential minimal optimization (SMO) algorithm [111]. The SVM classifier can be used to determine non-linear discriminant hyperplanes via a technique called kernel trick, which implicitly maps the input features to higher dimensional spaces. However, to keep the model simple and improve its generalizability, we utilize a lin-

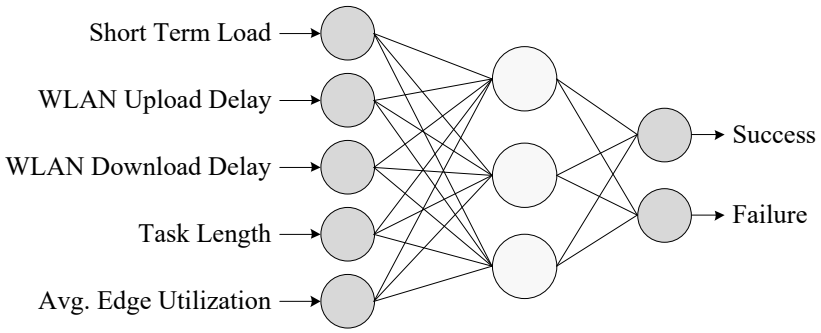


Figure 6.4. MLP neural network of the *edge* classifier.

ear kernel. The C parameter (regularization parameter, which determines the model's complexity) is selected to be 1.

MLP is an artificial neural network which has an input/output layer and an arbitrary number of hidden layers between them [112]. The hidden layers are composed of a multitude of neurons that implement the sigmoid activation function. Hence they can model non-linear relationships between the target classes and input features. The connections between neurons are assigned weights that are learned via the backpropagation algorithm, which is based on the gradient descent algorithm. The learning rate parameter determines the amount of model change allowed on each epoch (iteration), while the momentum parameter helps the algorithm to avoid converging on local minima. The settings of the MLP neural network used for "edge", "cloud via RSU" and "cloud via cellular" cases are similar. We use an MLP model with a fully connected single hidden layer, i.e., the neurons on the hidden layer are connected to all the neurons on the input and the output layers. As an example, the MLP neural network of the "edge" classifier is depicted in Figure 6.4. The specific parameters of the utilized MLP models are given in Table 6.1.

We inspected the true positive (T_p), true negative (T_n) rates, and their weighted average values to compare the classification algorithms. The classifiers used for "cloud via RSU" and "cloud via cellular" scenarios have high precision, but the classifier used for "edge" has lower precision since predicting the result of tasks offloaded to the VEC

Table 6.1. MLP Model Parameters.

Parameter	**Value**
Number of hidden units	2 and 3
Learning rate for the back-propagation algorithm	0.1
Momentum rate for the back-propagation algorithm	0.2
Number of epochs to train through	1000

server is the most complex problem. The performance comparison of the classifiers used for "edge" is shown in Figure 6.5. SVM and NB provide higher T_p rate, but the T_n rate is not satisfactory. These classifiers predict that the offloading result will be frequently successful. MLP gives more balanced results than its competitors, and it has a higher percentage of correctly classified instances. Therefore, we use the MLP based classifier as the first stage of our ML-based edge orchestrator.

The distribution of the training data is essential for the training phase. The training data set includes more entries for the successfully offloaded tasks. If all the collected data is used, the classifiers tend to predict "success", which yields to a poor

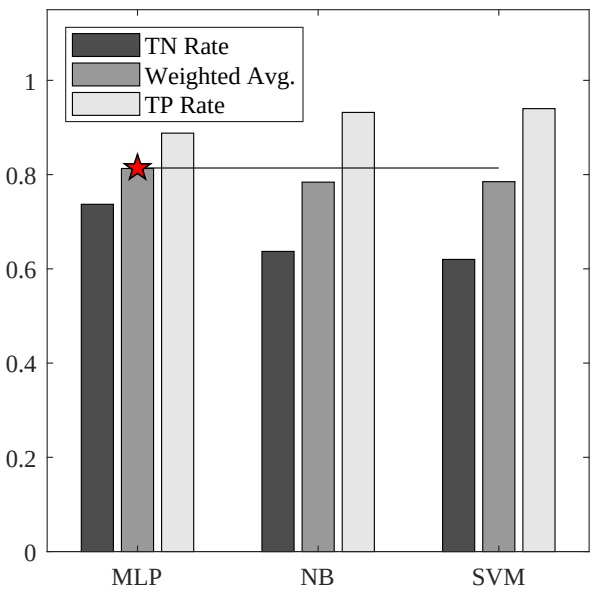


Figure 6.5. Performance comparison of the classifiers used for the edge scenario.

Table 6.2. Comparison of training data sets used by the *edge* classifier.

Data Set	T_p	T_n	**Recall**	**Precision**	**F-Score**
unbalanced	0.999	0.017	0.905	0.877	0.863
balanced	0.763	0.850	0.806	0.809	0.806
with VD	0.750	0.869	0.810	0.814	0.809
with STL	0.737	0.888	0.813	0.820	0.812
VD + STL	0.745	0.878	0.812	0.817	0.811

T_n rate. Our original training data set includes around 35 million samples, but around 2 million of them belong to the failed tasks. The training data is rearranged randomly in a way to use an equal number of "success" and "failure" cases. This process reduced the number of samples to around 4 million, which includes 250 thousand, 750 thousand, and 1 million samples of failed tasks for the *edge*, *cloud via RSU* and *cloud via CN* scenarios respectively. Balancing the number of samples for the success and failure cases improves the T_n rate significantly, as shown in Table 6.2. After the training data was balanced, the best data set whose elements have the strongest relationship between the inputs and output (correlation coefficient) are found by testing the different sets. We have selected some features based on our previous work and knowledge in this field. For example, the WLAN status, the task length, and the average utilization of edge servers have a direct impact on the outcome of offloading to RSUs. On the other hand, the computational parameters do not affect the result of cloud offloading since the cloud servers usually have enough computing resources. Therefore, the network resources such as the WAN and cellular network capacity are taken into account while offloading to the cloud.

In addition to experience-based features, we tried to improve the prediction performance by using additional features, which may have a positive impact on the results. Since the classifiers used for the edge has lower precision than the others, we worked on this model more intensively. The short-term load (STL) and vehicle density (VD) are tested as new features. STL refers to the number of tasks offloaded to the related

server in the recent past, such as the last 5 seconds. The load of the servers does not change immediately after making the decision because the offloaded task reaches to the destination after a network delay. So, the STL value can infer a possible load in the future. VD is the number of vehicles used in the simulation. The system can behave differently under different loads, so this value is used to analyze whether it has an impact on the classifier performance. The important performance metrics for different data sets are listed in Table 6.2. Adding the STL and VD features to the training data set also increases the classifier performance. We use the F-Score value, which is commonly used to compare the classifiers. The recall and precision values are used to compute the F-Score value. Related metrics are calculated by

$$precision = (\frac{T_p}{T_p + F_p} + \frac{T_n}{T_n + F_n})/2 \tag{6.1}$$

$$recall = (\frac{T_p}{T_p + F_n} + \frac{T_n}{T_n + F_p})/2 \tag{6.2}$$

$$f - score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{6.3}$$

Using STL and VD features separately or together provide similar performances, but the "only STL" case provides slightly better F-Score. In addition, finding the density of vehicles on the road may not be easy in real-life scenarios. Therefore, we chose using the balanced data set with the STL feature to prevent the overhead of calculating VD. The list of features used for each classifier is given in Table 6.3.

Table 6.3. List of features used for the classification models.

Classifier	**Feature 1**	**Feature 2** $*$	**Feature 3**	**Feature 4**
edge	STL	WLAN delay	task length	avg. edge util.
cloud via RSU	STL	WAN delay	-	-
cloud via CN	STL	CN delay	-	-

$*$ This feature refers two features for up-link and down-link delays.

Table 6.4. List of features used for the regression models.

Regressor	**Feature 1**	**Feature 2**	**Feature 3**
edge	task length	Avg. edge util.	-
cloud via RSU	task length	WAN upload delay	WAN download delay
cloud via CN	task length	CN upload delay	CN download delay

6.1.4. Regression Models

The regression model is used to predict the service time of the offloaded task. A linear regression model is utilized to predict the service time because the output variable (service time) is a linear combination of our inputs (features). We applied a similar study as we did for the classification models to find the best feature list for the regression models. The list of features used for each regression model is given in Table 6.4.

6.2. Performance Evaluation

The performance of the proposed ML-based workload orchestrator is evaluated through an extensive number of experiments. The simulations are performed on EdgeCloudSim, which is capable of modeling both the computational and networking resources as well as the mobile vehicles [73].

6.2.1. Competitor Algorithms

In order to benchmark the proposed solution, we implement three competitor algorithms, which are the *random*, *simple moving average* (SMA) based, MAB theory-based, and game theory-based vehicular edge orchestrators. The random orchestrator selects the target server to offload in a probabilistic manner where the probability of selecting all targets is the same. This strategy is used to present the worst-case scenario. SMA is a time series forecasting method that uses short-term historical data

for all $t \in T$ **do** ▷ For all Tasks

  for all $c \in C$ **do** ▷ For all Candidates

    $SCR_c \leftarrow$ GETSCORE(c)

  end for

  Offload the task to candidate c_t , such that:

  $c_t \leftarrow \max_{\forall c \in C} SCR_c$

  Update statistic list A

end for

procedure GETSCORE(c)

  $totalScore \leftarrow 0$

  for all i in A **do**

    $weight \leftarrow L(A) - I(A_i)$

    $score \leftarrow SR_{c,i} * weight$

    $totalScore \leftarrow totalScore + score$

  end for

  Return $totalScore$

end procedure

Figure 6.6. SMA based vehicular edge orchestrator.

to estimate future outcomes. This algorithm uses a fixed-length list to record the time series data, wherein each element of the list contains the data for a specific period. It calculates the average success rate of each option by using a higher weight for the data, which is collected more recently. Then it selects the target machine that has given the highest success rate. The pseudocode of the SMA based workload orchestrator is given in Figure 6.6 where T is the incoming task array, C refers to the candidate processing unit array, c_t is the candidate processing unit for task t, SCR_c is the preference score of candidate processing unit c, A is the list of statistical data, $L(A)$ is the length of list A, $I(A_i)$ is the index of element i of list A, and $SR_{c,i}$ is the average success rate of candidate c for element (time period) i of list A.

MAB is a reinforcement learning approach that tries to maximize the expected gain. We implemented a MAB theory-based orchestrator for our multi-access multi-tier VEC architecture based on Sun et al.'s work [61]. In their solution, the candidate machines to offload are the neighboring vehicles, while in our solution, the candidates are the edge servers and cloud server. In addition, the cloud server can be accessed via two ways in the proposed architecture, one is the WAN accessed through RSUs, and the other one is the cellular network. Therefore, we have three candidates for the task offloading, which are "edge", "cloud via RSU" and "cloud via CN" as explained in Section 6.1. Although the candidates seem to be different, the problems solved in both studies are very similar.

The strategy used to solve the MAB problems varies. The upper confidence bound (UCB) algorithm is one of the most widely used strategies. Sun et al. use a UCB strategy with a slight change. They added the input-awareness and occurrence-awareness to the UCB formula and named it as the "adaptive volatile UCB" (AVUCB). We use the same formula without the occurrence-awareness as given in Figure 6.7 where t refers to the time, $d_{sum}(t, n)$ is the service time of candidate n at time t, x_t is the length of task, $\tilde{x_t}$ is the normalized task length, $u_{t,n}$ is the sum delay of offloading one instruction, $k_{t,n}$ is the number of tasks that have been offloaded to candidate n up till time t, τ is a constant factor, used 1 in the simulations, and a_t is the candidate with the minimum utility value.

The input-awareness is used to eliminate the imbalance utility function score of the small and large tasks. Equation 6.5 in Figure 6.7 uses x_t to achieve the input-awareness. According to our observations, the input-awareness significantly improves the orchestrator performance. The occurrence-awareness is used to provide equal chances to the serving vehicles. Since the neighbors of the vehicles can frequently change due to the mobility, AVUCB uses a different time value for each serving vehicle. However, in our architecture, vehicles always have three candidates, so this concept is not necessary for us. We use $ln(t)$ in Equation 6.4 of Figure 6.7, while AVUCB uses $ln(t - t_n)$ where t_n is the occurrence time of each serving vehicle.

for all $t \in T$ **do**

if Any candidate $n \in N$ has not selected yet **then**

Select Candidate n

$\bar{u}_{t,n} \leftarrow d_{sum}(t,n)/x_t$

$k_{t,n} \leftarrow 1$

else

Observe x_t, calculate $\tilde{x_t}$

Calculate utility func. of each candidate $n \in N$

$$\tilde{u}_{t,n} \leftarrow \bar{u}_{t-1,n} - \sqrt{\frac{\tau(1-\tilde{x_t})ln(t)}{k_{t-1,n}}} \tag{6.4}$$

Offload the task to candidate a_t , such that:

$a_t \leftarrow \underset{n \in N}{argmin}\ \tilde{u}_{t,n}$

Observe the sum offloading delay $d_{sum}(t,n)$

$$\bar{u}_{t,a_t} \leftarrow \frac{\bar{u}_{t-1,a_t} k_{t-1,a_t} + d_{sum}(t,n)/x_t}{k_{t-1,a_t}+1} \tag{6.5}$$

$k_{t-1,a_t} \leftarrow k_{t-1,a_t} + 1$

end if

end for

Figure 6.7. MAB theory based vehicular edge orchestrator.

The game theory-based algorithm uses a multi-user non-cooperative computation offloading game to adjust the offloading probability of vehicles to achieve the maximum utility. We implemented the computation offloading game based on Wang et al.'s work [62]. They use a best-response strategy, which is given in Equation 6.8. This strategy uses the latest offloading probabilities of other vehicles to converge to a unique Nash equilibrium. In Wang et al. model, the tasks can be executed on the vehicles locally or on the edge servers remotely. They try to find the probability of offloading to edge servers. However, our model has three remote offloading options, as explained in Section 6.1. In order to use their game model, we decrease our options as edge or

Initialization: offloading probability vector p, pricing vector ρ, and task arrival rate vector λ.

for all $t \in T$ **do**

$i \leftarrow VehicleIndex$ ▷ t generated by i^{th} vehicle

Estimate transmission delay $d^t_{i,e}$ and $d^t_{i,c}$ of task for WLAN and WAN

Estimate processing delay $d^c_{i,e}$ and $d^c_{i,c}$ of task on edge and cloud:

$$d^p_{i,e} \leftarrow \frac{x_t}{C_e}(1 - U_e) \tag{6.6}$$

$$d^p_{i,c} \leftarrow \frac{x_t}{C_c}(1 - U_c) \tag{6.7}$$

Calculate total delay $d_{i,e}$ and $d_{i,c}$ of task on edge and cloud:

$d_{i,e} \leftarrow d^p_{i,e} + d^t_{i,e}$

$d_{i,c} \leftarrow d^p_{i,c} + d^t_{i,c}$

Update the best-response strategy:

$$p_i = \left[\frac{d_{i,e} - d_{i,c}}{2\rho d_{i,max}(1 - \prod_{i \neq j}(1 - \lambda_j p_j))} \right]^1_0 \tag{6.8}$$

end for

Figure 6.8. Game theory-based vehicular edge orchestrator.

cloud offloading. If the model decides to execute a task on the cloud server, we prefer either the cellular network or high-speed network provided by the RSU according to the bandwidth they promised. The modified game theory-based vehicular edge orchestrator algorithm is given in Figure 6.8 where x_t is the length of the task, C_e and C_c are the capacity of edge and cloud servers, U_e and U_c are the average utilization of edge and cloud servers between 0 and 1, $d_{i,max}$ is the maximum delay for the task generated by vehicle i, $d_{i,e}$ is the estimated delay on edge for the task generated by vehicle i, $d_{i,c}$ is the estimated delay on cloud for the task generated by vehicle i, ρ is the pricing factor, λ_i is the task arrival rate of vehicle i, and p_i is the probability of offloading to cloud

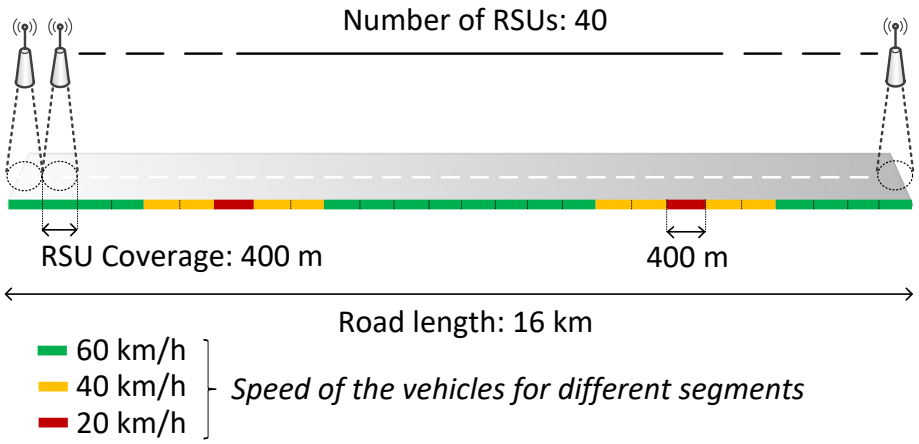


Figure 6.9. Vehicular mobility model in the simulation.

for the task generated by vehicle i. The pricing factor ρ is used to adjust the degree of willingness to cloud offloading. A smaller pricing factor increases the possibility of uploading to the cloud server. In our simulations, the pricing factor is selected as 0.6. The task arrival rate is utilized as $\lambda \in [0.5, 0.9]$ in the original work. However, this value varies between 3 and 15 in our experiments. Therefore, we normalize λ between 0.15 and 0.75 to avoid p_i of going to positive or negative infinity.

6.2.2. Simulation Setup

The road and mobility model used in the simulations is implemented on EdgeCloudSim, as shown in Figure 6.9. To simulate the vehicular mobility more realistically, the road is divided into segments, and a dynamic velocity value for the vehicle position is used. Therefore, the speed of the vehicles varies at each segment to differentiate the traffic density on the road. The hotspot locations, which are shown with red color, occur due to the traffic jam. 100 to 1800 vehicles are distributed to random locations when the simulation is started. Then they move in a single direction with a predefined speed with respect to the crossed segment. The road is modeled as a circular route to keep the number of vehicles the same during the simulation.

The vehicles offload some of their tasks to the other computing units. It is assumed that the vehicles handle some operations locally, but some tasks are offloaded to a remote server if the OBU is not capable of executing the related tasks. The workload orchestrator takes care of the tasks which are decided to be offloaded to a remote server, which can be an edge or a cloud server. Therefore, the OBU execution is not addressed in this study. The edge servers are provided by the RSUs, which also operate an access point. The transmission between the vehicles and RSUs is performed over an 802.11p WLAN. According to Lin et al.'s measurements, the 802.11p protocol can achieve around 10 Mbps data rate using 16QAM for the V2I communication [113]. Therefore, we use a 10 Mbps data rate for the 802.11p interface. All RSUs are connected to a high-speed network with Internet access that can be provided through conventional infrastructures such as a fiber network. The vehicles can offload their tasks to the cloud server through the RSU or the cellular network provided by the LTE base station. Xu et al. carried out the experiments to evaluate the performance of 4G-LTE used by the connected vehicles [114]. Based on their observations, we use a 20 Mbps data rate for the LTE cellular network.

We use three different applications that run on the vehicles to generate tasks

Table 6.5. Application characteristics.

	Navigation App	**Danger Assessment**	**Infotainment App**
Usage Ratio∗ (%)	30	35	35
Task Interarrival time (sec)	3	5	15
Max Delay Requirement (sec)	0.5	1	1.5
Delay Sensitivity (sec)	0.5	0.8	0.25
Upload/Download Data (KB)	20/20	40/20	20/80
Task Length (GI)	3	10	20
RSU/Cloud VM Utilization(%)	6/1.6	20/4	40/8

∗ Percentage of the vehicles requesting related service.

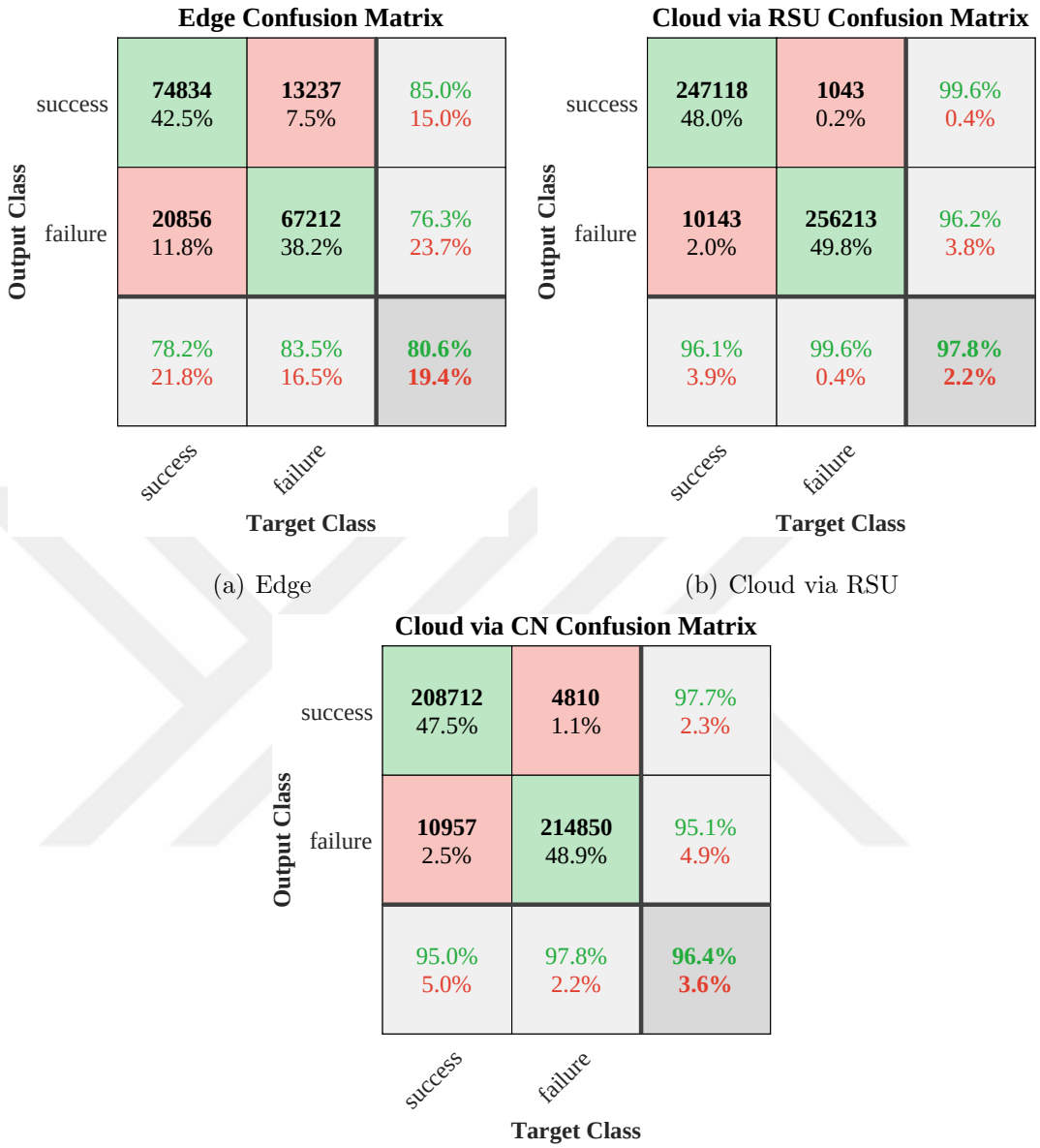
Table 6.6. Simulation parameters.

Parameter	**Value**
Simulation Time/Warm-up Period	60/3 minutes
Number of Repetitions	100
Network Delay Model	MMPP/M/1 queue model
# of VMs per Cloud/RSU	20/40 (2 per RSU)
Capacity of Cloud/RSU VM	150/20 GIPS
Range of RSU (WLAN)	200 meters
WLAN/MAN Bandwidth	10/1000 Mbps
WAN/WAN over LTE Bandwidth	50/20 Mbps
WAN/WAN over LTE Propagation Delay	150/160 ms

at different rates. The applications have different characteristics, such as the inter-arrival times of tasks, the length of tasks, and the data size to upload or download. The characteristics of the applications are given in Table 6.5, and the other simulation parameters are listed in Table 6.6.

6.2.3. Simulation Results and Discussions

Before presenting the comparative evaluation of the algorithms, the performance of the classifiers used in the first stage of our ML-based workload orchestrator is addressed. The confusion matrix of each classifier is shown in Figure 6.10. The green boxes in the confusion matrix correspond to correctly classified observations, while the red boxes show the misclassified instances. The columns on the far right of the plot show the precision for each class. The rows at the bottom of the plot show the recall for each class. The cell in the bottom right of the plot shows the overall accuracy. As already stated, it is more difficult to predict the result of a task offloaded to the edge server. Therefore, the MLP based classifier for edge provides the lowest accuracy while it is adequate for the other cases.



(a) Edge

(b) Cloud via RSU

(c) Cloud via CN

Figure 6.10. Confusion matrix of the classifiers.

The failure rate is one of the most critical performance criteria. A task cannot be executed if the utilization of a VM is too high to accept the offloaded tasks, or if the network bandwidth is not sufficient to transfer either the input or the output of the task in a reasonable amount of time. Therefore, a task can fail due to the lack of computational or networking resources in our simulation. The mobility also leads to the task failure, but this case is not addressed since it accounts for a small proportion of the total unsuccessful operations. The average number of failed tasks with respect to

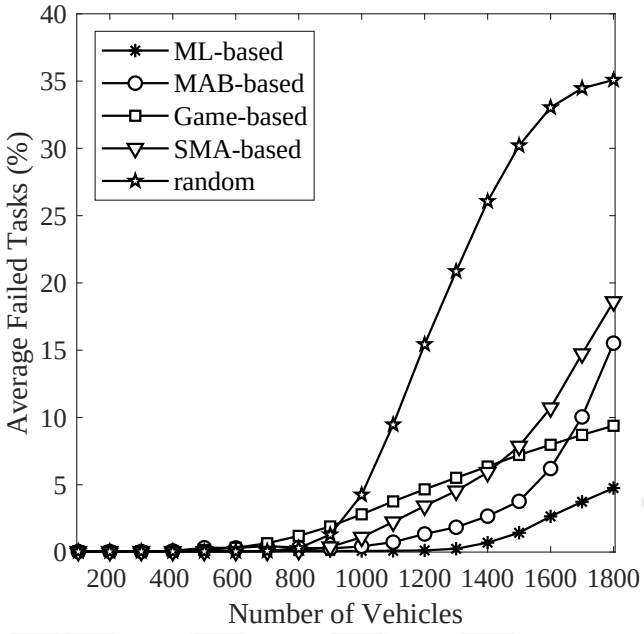
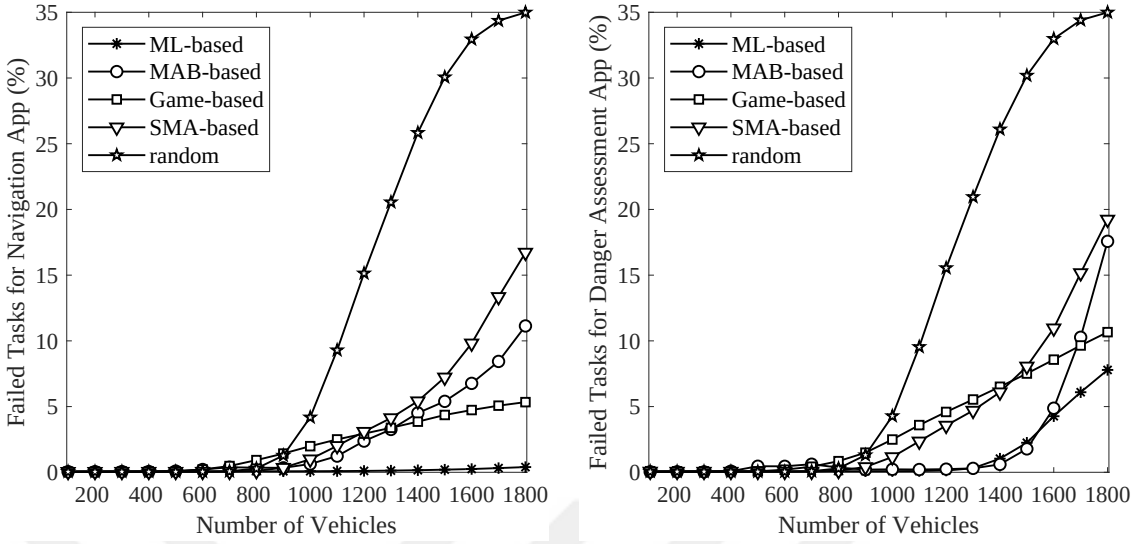


Figure 6.11. Percentage of failed tasks based on all application types.

the number of vehicles is shown in Figure 6.11. The ML-based approach outperforms its competitors because the other approaches have an exploration phase in which some wrong decisions are made to explore the current situation of the system and make more accurate decisions accordingly. Especially after 1500 vehicles, the system is overloaded, and even a small number of wrong decisions adversely affect the performance. The ML-based solution provides superior results for the small tasks which are generated by the navigation application. The inter-arrival time of the small tasks is shorter than the other tasks, so the navigation application has more impact on the average failed task performance.

The results for small and relatively large task sizes are shown in Figure 6.12. Figure 6.12(a) presents the percentage of failed tasks for the navigation application, while Figure 6.12(b) presents the same results for the danger assessment application. According to our observations, the ML-based algorithm tends to send the small tasks to a nearby edge server and the larger to a remote cloud server. The MAB-based algorithm is also a load-aware solution that takes care of the task size. However, it



(a) Navigation app. score

(b) Danger assessment app. score

Figure 6.12. Average failed tasks for different application types.

does not work well when the system becomes congested, and consequently, the loss of tasks begins. The game theory-based algorithm offloads more of the task to edge servers regardless of the task type. The best-response strategy used in the game model uses the predicted service time values as the input variables and does not have any feedback mechanism to evaluate its decision. Almost all of the task failures occur due to the lack of the edge server capacity. Therefore, the task loss situation is observed after 600 vehicles and increases linearly. The SMA-based algorithm does not consider the task size. In particular, it increases the probability of selecting a computing unit that has recently yielded better results. Hence, this strategy results in more task loss when the system is overloaded.

The average service time score of the successfully offloaded tasks can mislead an evaluator in the delay-loss systems. For example, we cannot say that the performance is good in a scenario where the orchestrator provides an adequate service time while losing most of the tasks. Therefore, we use a Quality of Experience (QoE) formula, which considers both the service time and task loss as defined in Equation 6.9.

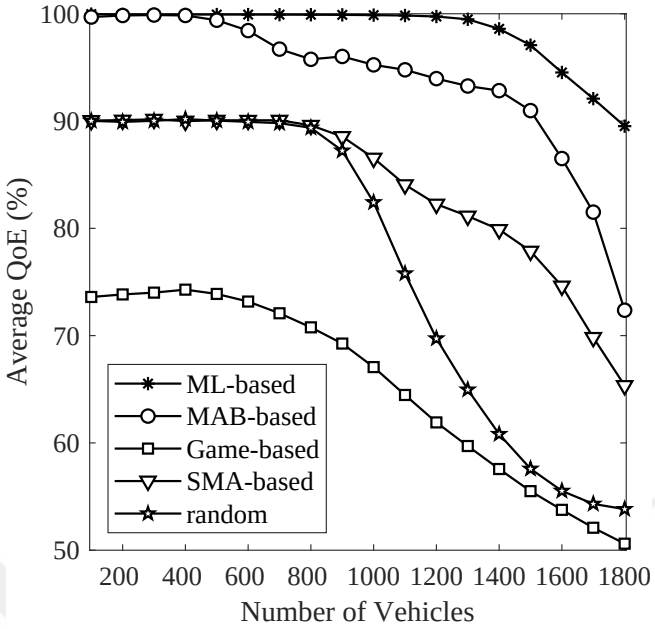


Figure 6.13. Average Quality of Experience based on all application types.

$$QoE_i = \begin{cases} 0, & \text{if } T_i \geq 2R_i \\ (1 - \frac{T_i - R_i}{R_i}) \cdot (1 - S_i), & \text{if } R_i < T_i < 2R_i \\ 1, & \text{if } T_i \leq R_i \end{cases} \tag{6.9}$$

where, T_i is the actual service time, R_i is the delay requirement, and S_i is the delay sensitivity of task i. The delay requirement is used to declare the maximum service time allowed for the related task. The average QoE value decreases when task i is completed later than R_i. If the service time exceeds twice the expected value, the QoE value is set to 0. The delay sensitivity refers to the delay tolerance. This value is between 0 and 1, and higher for the delay intolerant applications. The average QoE with respect to the number of vehicles is shown in Figure 6.13. We determined the delay requirements (in proportion to the task sizes) and delay sensitivities of tasks as in Table 6.5. The ML-based and MAB-based approaches provide the maximum QoE (100%) when the number of vehicles is low, e.g. less than 500, because their goal is to

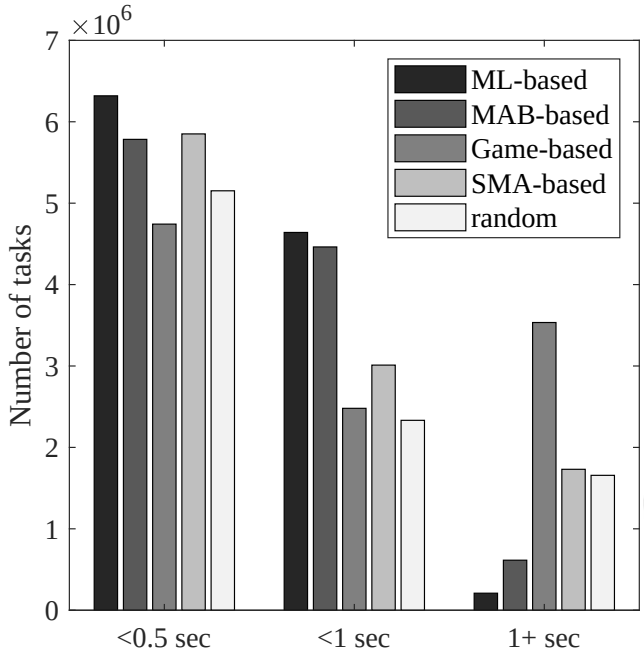
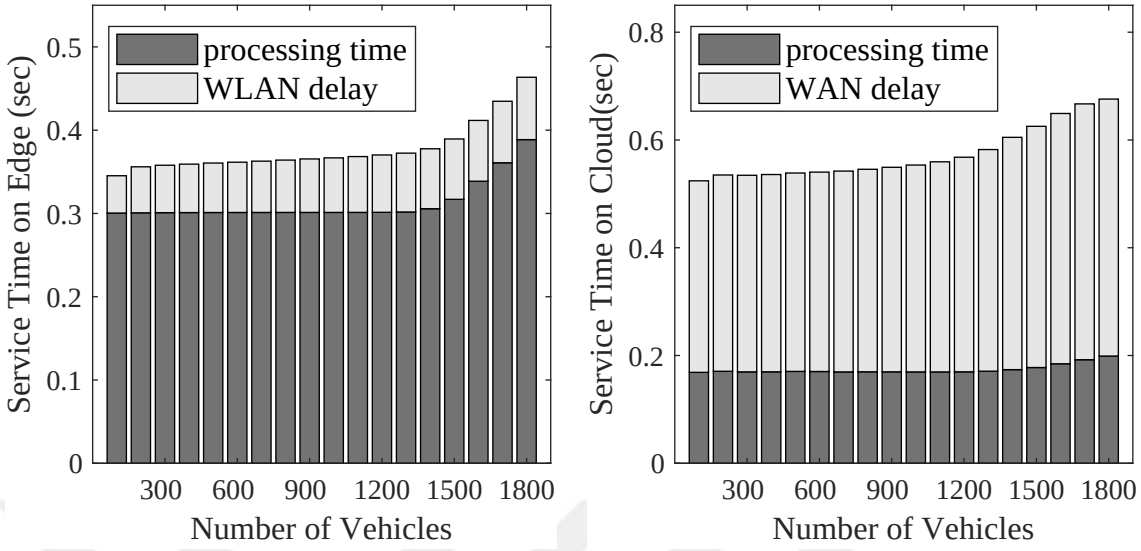


Figure 6.14. Histogram of service times based on all application types.

minimize the service time. However, the random and SMA-based approaches do not aim to minimize the service time, so their maximum QoE is around 90% even if there is no task failure. In general, the ML-based method provides the best performance in terms of QoE. The MAB-based solution does not meet the delay requirement criteria after the number of vehicles exceeds 500. The main reason for this problem is that the MAB-based approach sends some large tasks to the edge servers and exceeds the maximum service time allowed for these tasks. On the other hand, the ML-based solution mostly sends large tasks to the cloud server and does not cause congestion in the edge layer. The Game-based solution provides the worst QoE performance because the main objective of the game model is not minimizing the service time; in fact, it tries to converge to a stable equilibrium.

The proportion of the service times of successfully completed tasks is shown in Figure 6.14. Since the smaller service time infers the better performance, it is better to have a higher value on the left side of the graph. The ML-based solution provides the best performance in terms of the service time because it has more tasks that are



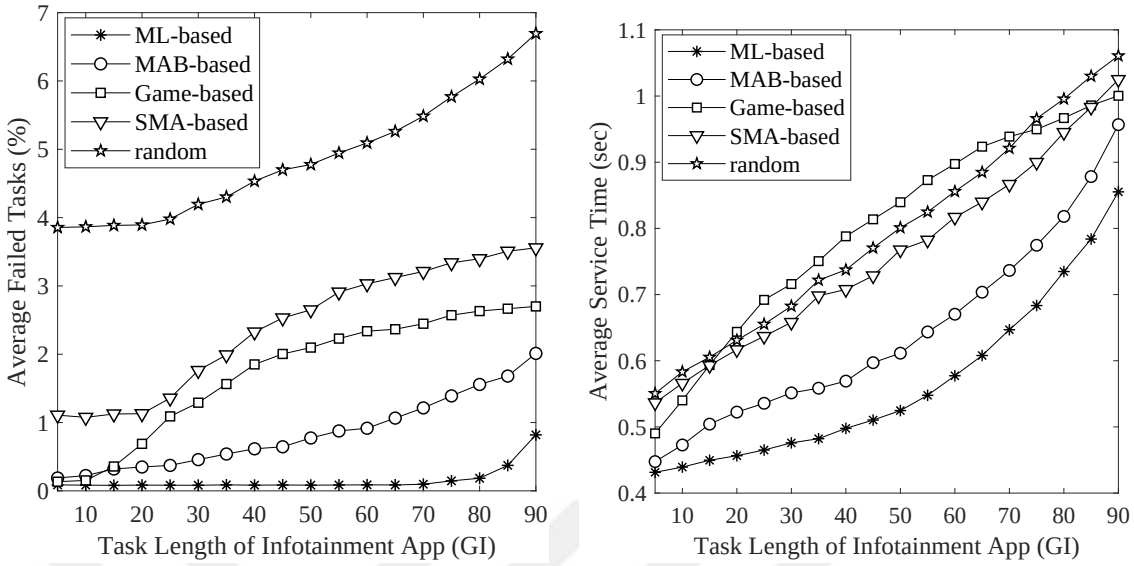
(a) Service times on edge servers

(b) Service times on cloud servers

Figure 6.15. Service time ratio of the edge and cloud servers.

completed under 0.5 seconds and fewer tasks that take more than 1.5 seconds to execute. With a similar perspective, the performance of the other algorithms is ranked from the best to the worst as the MAB-based, SMA-based, random, and Game-based. Since the Game-based model offloads more of the task to the edge server, it causes significant congestion on edge servers. The service time includes both the computational and networking delay. The ratio of these delays is shown in Figure 6.15. Figure 6.15(a) gives the service time ratio of edge servers, whereas Figure 6.15(b) presents the same result for the cloud servers. As expected, the significant proportion of the service times on the edge servers occur due to the processing delay. On the other hand, cloud servers execute tasks in a fast manner, but it takes more time to transmit the data to the cloud servers.

We increase the load on the system by altering the number of vehicles used in the simulation. The load can also be generated by other methods, such as increasing the task arrival rate and task length. As an example, we performed an experiment by changing the task length of the infotainment application. The task length of this application was 20 GIPS in our detailed experiments. We changed this value between 5 and 90 GIPS in a scenario with 1000 vehicles. The average task failure and the service



(a) Average failed tasks (%)

(b) Average service time (sec)

Figure 6.16. Comparative evaluation for different task length.

time performances are shown in Figure 6.16. It can be seen that the performance decreases as the task length increases as expected. The proposed ML-based approach again provides the best score in this experiment.

7. CONCLUSION

In this thesis, we focused on the workload orchestration for edge computing systems and the performance evaluation of different edge orchestrators. We also spent a considerable amount of time on the modeling approaches of these systems in order to simulate our scenarios more efficiently and realistically. As a result of this study, we introduced our open-source edge computing simulator for education and research.

Edge computing is a fast-growing field of research that covers a spectrum of technologies bringing the cloud computing resources closer to the end-user. The main objective of this paradigm is to improve the responsiveness and battery lifetime of mobile devices by processing the complex operations on the nearby micro-cloud servers located at the edge of the network instead of locally on the mobile devices or the central cloud. Some examples of the complex operations which cannot be handled on the mobile devices properly are; image processing, voice to text conversion, and big data processing. In the near future, mobile users would decide where to handle service-oriented demands of a wearable or sensor like IoT devices according to the computational resource, network, and battery conditions. These demands can be handled on different tiers such as the proximity edge server, intermediate or global cloud tiers. The number of tiers and the architecture of the systems may be different according to the targeted use cases.

The edge computing concept creates new opportunities and use-cases for mobile devices with limited capacity. This phenomenon has attracted the researchers' attention in recent years. However, the difficulty of simulating a realistic edge computing environment is one of the considerable obstacles the researches have faced. Some cloud computing simulators can handle provisioning of the computing elements such as datacenter, hosts, VMs efficiently, but they do not provide a practical method for modeling the network and mobility. Conventional network simulators are very effective in modeling the network, but they do not cover cloud computing elements. As a

result, evaluating the performance of different edge computing scenarios by considering mobility, network, and computational resources is a crucial requirement. To meet this requirement and assist other researchers, we have developed a simulator called EdgeCloudSim which is publicly available in GitHub [9]. We also published a discussion group [115] with 84 members and a YouTube channel [116] group with some help videos. EdgeCloudSim is an accepted edge computing simulator now. It has become a popular tool that is cited by more than 120 publications as of May 2020. We released two major versions up to now and are still making an effort to add the new features to make our simulator more powerful. Currently, we are working on adding the vehicular mobility model that is compatible with the SUMO traffic simulation tool.

Although the edge computing philosophy provides a clear advantage for the delay intolerant applications or services, the limited nearby resources should be managed carefully. The dynamic state of the network, mobile devices, and edge servers brings some challenges which negatively affects the overall system performance. Since the demand from mobile devices is not known in advance in such a dynamic environment, the resource management becomes a hard online problem. In this manner, the edge orchestration poses a crucial problem. In essence, the edge orchestrator has significant responsibilities such as the resource allocation on VMs, memory, CPU provisioning on servers, power management on datacenters, and workload orchestration. In this thesis, we concentrated on the workload orchestration, which is basically deciding how to distribute the offloading requests. Deciding where to offload an incoming task is a difficult problem that cannot be solved in polynomial time. Therefore, we use different online problem-solving techniques to overcome this complexity. Firstly, we propose a novel fuzzy logic-based workload orchestrator for multi-tier edge computing systems due to its low complexity and efficiency in handling uncertain nonlinear systems by considering multiple criteria. The mobile devices offload their tasks to the edge or cloud servers via WiFi access technologies, and the orchestrator allocates the edge resources. We set up a simulation environment to evaluate the performance of our workload orchestrator design by comparing it with the classical workload orchestrator algorithms. According to the simulation outcomes, using a fuzzy logic-based orchestrator provides better re-

sults compared to the other algorithms for the cases studied. As future work, we plan to apply intelligent optimization techniques to improve our fuzzy membership functions and rules. In addition, we want to compare our FLS with other techniques, such as using a reinforcement learning approach for workload orchestration [117]. Finally, it can be interesting to analyze the impact of task migration among the edge/cloud servers.

As the next step in our study, we concentrated on the VEC domain. The main objective of VEC is to bring the cloud resources in close proximity to the vehicles. The services needed on the smart roads will increase, as the number of IoV applications increases. These services are expected to provide many emerging smart vehicular applications that require significant computing power. The VEC promises to handle computation-intensive and latency-sensitive tasks generated by the IoV applications. Orchestrating the dynamic and heterogeneous resources in the VEC systems is a challenging task. The vehicular workload orchestrator promises to offload the incoming tasks (workload) to the optimal computing unit to improve the system performance. The conventional solution techniques, such as offline optimization methods cannot solve this problem efficiently. We propose an ML-based workload orchestrator for multi-access multi-tier VEC to solve this problem. It can handle the uncertain nonlinear systems efficiently by considering multiple criteria. We also implemented an SMA based and MAB theory-based approaches to compare the proposed solution. Based on the extensive simulation results, the ML-based workload orchestrator outperforms its competitors in terms of the average task failure rates and QoE which includes the service time. As a future work, we plan to optimize the features used in the classification and regression stages of the ML-based workload orchestrator by using an intelligent optimization algorithm. In addition, we want to implement a deep learning-based approach to solve the workload orchestration problem. Finally, we want to enhance our problem by enabling a V2V offloading in which the vehicles have an on-board computation unit.

REFERENCES

1. Taleb, T., K. Samdanis, B. Mada, H. Flinck, S. Dutta and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration", *IEEE Communications Surveys Tutorials*, Vol. 19, No. 3, pp. 1657–1681, thirdquarter 2017.

2. Satyanarayanan, M., P. Bahl, R. Caceres and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing", *Pervasive Computing, IEEE*, Vol. 8, No. 4, pp. 14–23, Oct 2009.

3. Dinh, H. T., C. Lee, D. Niyato and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches", *Wireless communications and mobile computing*, Vol. 13, No. 18, pp. 1587–1611, 2013.

4. Cisco, *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*, 2015, `https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf`, accessed in May 2020.

5. ETSI, *Multi-access Edge Computing*, 2017, `http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing`, accessed in May 2020.

6. Orsini, G., D. Bade and W. Lamersdorf, "CloudAware: Empowering context-aware self-adaptation for mobile applications", *Transactions on Emerging Telecommunications Technologies*, Vol. 29, No. 4, p. e3210, Aug 2017.

7. Flores, H., X. Su, V. Kostakos, A. Y. Ding, P. Nurmi, S. Tarkoma, P. Hui and Y. Li, "Large-scale offloading in the Internet of Things", *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 479–484, March 2017.

8. Sonmez, C., A. Ozgovde and C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of edge computing systems", *Transactions on Emerging Telecommunications Technologies*, Vol. 29, No. 11, p. e3493, 2018.

9. Sonmez, C., *EdgeCloudSim GitHub Page*, 2017, `https://github.com/CagataySonmez/EdgeCloudSim`, accessed in May 2020.

10. Sonmez, C., A. Ozgovde and C. Ersoy, "Fuzzy Workload Orchestration for Edge Computing", *IEEE Transactions on Network and Service Management*, Vol. 16, No. 2, pp. 769–782, 2019.

11. Sonmez, C., C. Tunca, A. Ozgovde and C. Ersoy, "Machine Learning Based Workload Orchestrator for Vehicular Edge Computing", *IEEE Transactions on Intelligent Transportation Systems*, 2020, submited.

12. Farris, I., T. Taleb, H. Flinck and A. Iera, "Providing ultra-short latency to user-centric 5G applications at the mobile network edge", *Transactions on Emerging Telecommunications Technologies*, Vol. 29, No. 4, p. e3169, March 2017.

13. Plakhteyev, A., A. Perepelitsyn and V. Frolov, "Edge computing for IoT: An educational case study", *IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pp. 130–133, May 2018.

14. Borthakur, D., H. Dubey, N. Constant, L. Mahler and K. Mankodiya, "Smart fog: Fog computing framework for unsupervised clustering analytics in wearable Internet of Things", *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 472–476, Nov 2017.

15. Cooney, N. J., K. P. Joshi and A. S. Minhas, "A Wearable Internet of Things Based System with Edge Computing for Real-Time Human Activity Tracking", *2018 5th Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, pp. 26–31, Dec 2018.

16. Google, *Google Glass*, 2014, `https://www.google.com/glass/start/`, accessed in May 2020.

17. MacHardy, Z., A. Khan, K. Obana and S. Iwashina, "V2X Access Technologies: Regulation, Research, and Remaining Challenges", *IEEE Communications Surveys Tutorials*, Vol. 20, No. 3, pp. 1858–1877, thirdquarter 2018.

18. Seo, H., K. Lee, S. Yasukawa, Y. Peng and P. Sartori, "LTE evolution for vehicle-to-everything services", *IEEE Communications Magazine*, Vol. 54, No. 6, pp. 22–28, June 2016.

19. Boban, M., A. Kousaridas, K. Manolakis, J. Eichinger and W. Xu, "Connected Roads of the Future: Use Cases, Requirements, and Design Considerations for Vehicle-to-Everything Communications", *IEEE Vehicular Technology Magazine*, Vol. 13, No. 3, pp. 110–123, Sep. 2018.

20. Peng, H., Le Liang, X. Shen and G. Y. Li, "Vehicular Communications: A Network Layer Perspective", *IEEE Transactions on Vehicular Technology*, Vol. 68, No. 2, pp. 1064–1078, Feb 2019.

21. Hou, X., Y. Li, M. Chen, D. Wu, D. Jin and S. Chen, "Vehicular Fog Computing: A Viewpoint of Vehicles as the Infrastructures", *IEEE Transactions on Vehicular Technology*, Vol. 65, No. 6, pp. 3860–3873, June 2016.

22. Xie, R., Q. Tang, Q. Wang, X. Liu, F. R. Yu and T. Huang, "Collaborative Vehicular Edge Computing Networks: Architecture Design and Research Challenges", *IEEE Access*, Vol. 7, pp. 178942–178952, 2019.

23. Zhang, K., Y. Mao, S. Leng, Y. He and Y. ZHANG, "Mobile-Edge Computing for Vehicular Networks: A Promising Network Paradigm with Predictive Off-Loading", *IEEE Vehicular Technology Magazine*, Vol. 12, No. 2, pp. 36–44, June 2017.

24. Zhou, H., W. Xu, J. Chen and W. Wang, "Evolutionary V2X Technologies Toward the Internet of Vehicles: Challenges and Opportunities", *Proceedings of the IEEE*, Vol. 108, No. 2, pp. 308–323, Feb 2020.

25. Li, X., M. Samaka, H. A. Chan, D. Bhamare, L. Gupta, C. Guo and R. Jain, "Network Slicing for 5G: Challenges and Opportunities", *IEEE Internet Computing*, Vol. 21, No. 5, pp. 20–27, 2017.

26. Afolabi, I., M. Bagaa, T. Taleb and H. Flinck, "End-to-end network slicing enabled through network function virtualization", *IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 30–35, Sept 2017.

27. Olariu, S., "A Survey of Vehicular Cloud Research: Trends, Applications and Challenges", *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–16, 2019.

28. Initiative, O. E. C., *Living Edge Lab*, 2020, `https://www.openedgecomputing.org/living-edge-lab/`, accessed in May 2020.

29. D'Angelo, G., S. Ferretti and V. Ghini, "Modeling the Internet of Things: a simulation perspective", *2017 International Conference on High Performance Computing Simulation (HPCS)*, pp. 18–27, July 2017.

30. Kecskemeti, G., G. Casale, D. N. Jha, J. Lyon and R. Ranjan, "Modelling and Simulation Challenges in Internet of Things", *IEEE Cloud Computing*, Vol. 4, No. 1, pp. 62–69, Jan 2017.

31. Foukas, X., G. Patounas, A. Elmokashfi and M. K. Marina, "Network Slicing in 5G: Survey and Challenges", *IEEE Communications Magazine*, Vol. 55, No. 5, pp. 94–100, May 2017.

32. Byrne, J., S. Svorobej, K. M. Giannoutakis, D. Tzovaras, P. J. Byrne, P.-O. Östberg, A. Gourinovitch and T. Lynn, "A Review of Cloud Computing Simula-

tion Platforms and Related Environments", *CLOSER*, April 2017.

33. Calheiros, R. N., R. Ranjan, A. Beloglazov, C. A. F. De Rose and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms", *Softw. Pract. Exper.*, Vol. 41, No. 1, pp. 23–50, Jan. 2011.

34. Kliazovich, D., P. Bouvry, Y. Audzevich and S. U. Khan, "GreenCloud: A Packet-Level Simulator of Energy-Aware Cloud Computing Data Centers", *IEEE Global Telecommunications Conference GLOBECOM*, pp. 1–5, Dec 2010.

35. Núñez, A., J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero and I. M. Llorente, "iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator", *J. Grid Comput.*, Vol. 10, No. 1, pp. 185–209, Mar. 2012.

36. Tian, W., Y. Zhao, M. Xu, Y. Zhong and X. Sun, "A Toolkit for Modeling and Simulation of Real-Time Virtual Machine Allocation in a Cloud Data Center", *IEEE Transactions on Automation Science and Engineering*, Vol. 12, No. 1, pp. 153–161, Jan 2015.

37. Brennand, C. A. R. L., J. M. Duarte and A. P. Silva, "SimGrid: A simulator of network monitoring topologies for peer-to-peer based computational grids", *2016 8th IEEE Latin-American Conference on Communications (LATINCOM)*, pp. 1–6, Nov 2016.

38. Makaratzis, A. T., K. M. Giannoutakis and D. Tzovaras, "Energy modeling in cloud simulation frameworks", *Future Generation Computer Systems*, 2017.

39. Amazon, *Amazon Elastic Compute Cloud (EC2)*, 2009, `https://aws.amazon.com/ec2/`, accessed in May 2020.

40. Microsoft, *Microsoft Azure*, 2014, `https://azure.microsoft.com`, accessed in May 2020.

41. Issariyakul, T. and E. Hossain, *Introduction to Network Simulator NS2*, Springer Publishing Company, Incorporated, 1 edn., 2008.

42. Varga, A. and R. Hornig, "An Overview of the OMNeT++ Simulation Environment", *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pp. 60:1–60:10, 2008.

43. Zeng, X., S. Kumar Garg, P. Strazdins, P. P. Jayaraman, D. Georgakopoulos and R. Ranjan, "IOTSim: a Cloud based Simulator for Analysing IoT Applications", *Journal of Systems Architecture*, 02 2016.

44. Sotiriadis, S., N. Bessis, E. Asimakopoulou and N. Mustafee, "Towards Simulating the Internet of Things", *28th International Conference on Advanced Information Networking and Applications Workshops*, pp. 444–448, May 2014.

45. Gupta, H., A. V. Dastjerdi, S. K. Ghosh and R. Buyya, "iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments", *arXiv preprint arXiv:1606.02007*, 2016.

46. Sotiriadis, S., N. Bessis, N. Antonopoulos and A. Anjum, "SimIC: Designing a New Inter-cloud Simulation Platform for Integrating Large-Scale Resource Management", *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 90–97, March 2013.

47. Mach, P. and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading", *IEEE Communications Surveys Tutorials*, Vol. 19, No. 3, pp. 1628–1656, thirdquarter 2017.

48. Hegyi, A., H. Flinck, I. Ketyko, P. Kuure, C. Nemes and L. Pinter, "Application Orchestration in Mobile Edge Cloud: Placing of IoT Applications to the

Edge", *IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pp. 230–235, Sept 2016.

49. Karagiannis, V. and A. Papageorgiou, "Network-integrated edge computing orchestrator for application placement", *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–5, Nov 2017.

50. Santoro, D., D. Zozin, D. Pizzolli, F. D. Pellegrini and S. Cretti, "Foggy: A Platform for Workload Orchestration in a Fog Computing Environment", *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 231–234, Dec 2017.

51. Baktir, A. C., A. Ozgovde and C. Ersoy, "Enabling service-centric networks for cloudlets using SDN", *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 344–352, May 2017.

52. Amento, B., B. Balasubramanian, R. J. Hall, K. Joshi, G. Jung and K. H. Purdy, "FocusStack: Orchestrating Edge Clouds Using Location-Based Focus of Attention", *IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 179–191, Oct 2016.

53. Pham, X.-Q. and E.-N. Huh, "Towards task scheduling in a cloud-fog computing system", *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–4, Oct 2016.

54. Simoens, P., L. V. Herzeele, F. Vandeputte and L. Vermoesen, "Challenges for orchestration and instance selection of composite services in distributed edge clouds", *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 1196–1201, May 2015.

55. Bittencourt, L. F., J. Diaz-Montes, R. Buyya, O. F. Rana and M. Parashar, "Mobility-Aware Application Scheduling in Fog Computing", *IEEE Cloud Computing*, Vol. 4, No. 2, pp. 26–35, March 2017.

56. Maio, V. D. and I. Brandic, "First Hop Mobile Offloading of DAG Computations", *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 83–92, May 2018.

57. Flores, H. and S. Srirama, "Adaptive Code Offloading for Mobile Cloud Applications: Exploiting Fuzzy Sets and Evidence-based Learning", *Proceeding of the Fourth ACM Workshop on Mobile Cloud Computing and Services*, MCS '13, pp. 9–16, ACM, New York, NY, USA, 2013.

58. Hosseini, S. M., M. Kazeminia, M. Mehrjoo and S. M. Barakati, "Fuzzy logic based mobile data offloading", *2015 23rd Iranian Conference on Electrical Engineering*, pp. 397–401, May 2015.

59. Cao, B., L. Zhang, Y. Li, D. Feng and W. Cao, "Intelligent Offloading in Multi-Access Edge Computing: A State-of-the-Art Review and Framework", *IEEE Communications Magazine*, Vol. 57, No. 3, pp. 56–62, March 2019.

60. Ning, Z., P. Dong, X. Wang, J. J. P. C. Rodrigues and F. Xia, "Deep Reinforcement Learning for Vehicular Edge Computing: An Intelligent Offloading System", *ACM Trans. Intell. Syst. Technol.*, Vol. 10, No. 6, Oct. 2019.

61. Sun, Y., X. Guo, J. Song, S. Zhou, Z. Jiang, X. Liu and Z. Niu, "Adaptive Learning-Based Task Offloading for Vehicular Edge Computing Systems", *IEEE Transactions on Vehicular Technology*, Vol. 68, No. 4, pp. 3061–3074, April 2019.

62. Wang, Y., P. Lang, D. Tian, J. Zhou, X. Duan, Y. Cao and D. Zhao, "A Game-based Computation Offloading Method in Vehicular Multi-access Edge Computing Networks", *IEEE Internet of Things Journal*, 2020.

63. Liu, P., J. Li and Z. Sun, "Matching-Based Task Offloading for Vehicular Edge Computing", *IEEE Access*, Vol. 7, pp. 27628–27640, 2019.

64. Wang, H., X. Li, H. Ji and H. Zhang, "Federated Offloading Scheme to Minimize

Latency in MEC-Enabled Vehicular Networks", *IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, 2018.

65. Xu, X., Y. Xue, X. Li, L. Qi and S. Wan, "A Computation Offloading Method for Edge Computing With Vehicle-to-Everything", *IEEE Access*, Vol. 7, pp. 131068–131077, 2019.

66. Dai, Y., 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, June 2019.

67. Krajzewicz, D., "Traffic simulation with SUMO–simulation of urban mobility", *Fundamentals of traffic simulation*, pp. 269–293, Springer, 2010.

68. Pacheco, J. and S. Hariri, "Anomaly behavior analysis for IoT sensors", *Transactions on Emerging Telecommunications Technologies*, Vol. 29, No. 4, p. e3188, May 2017.

69. García-Pérez, C. A. and P. Merino, "Experimental evaluation of fog computing techniques to reduce latency in LTE networks", *Transactions on Emerging Telecommunications Technologies*, Vol. 29, No. 4, p. e3201, June 2017.

70. Yaseen, Q., F. Albalas, Y. Jararwah and M. Al-Ayyoub, "Leveraging fog computing and software defined systems for selective forwarding attacks detection in mobile wireless sensor networks", *Transactions on Emerging Telecommunications Technologies*, Vol. 29, No. 4, p. e3183, May 2017.

71. D'Angelo, G., S. Ferretti and V. Ghini, "Modeling the Internet of Things: a simulation perspective", *2017 International Conference on High Performance Computing Simulation (HPCS)*, pp. 18–27, July 2017.

72. Riverbed, *Riverbed Modeler - (formerly OPNET)*, 2012, `https://www.riverbed.com/gb/products/steelcentral/steelcentral-riverbed-modeler.html`, ac-

cessed in May 2020.

73. Sonmez, C., A. Ozgovde and C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of Edge Computing systems", *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 39–44, May 2017.

74. Kecskemeti, G., G. Casale, D. N. Jha, J. Lyon and R. Ranjan, "Modelling and Simulation Challenges in Internet of Things", *IEEE Cloud Computing*, Vol. 4, No. 1, pp. 62–69, Jan 2017.

75. Silva, M., D. Freitas, E. Neto, C. Lins, V. Teichrieb and J. M. Teixeira, "Glassist: Using Augmented Reality on Google Glass as an Aid to Classroom Management", *XVI Symposium on Virtual and Augmented Reality*, pp. 37–44, May 2014.

76. Guo, J., B. Song, Y. He, F. R. Yu and M. Sookhak, "A Survey on Compressed Sensing in Vehicular Infotainment Systems", *IEEE Communications Surveys Tutorials*, Vol. PP, No. 99, 2017.

77. Tunca, C., N. Pehlivan, N. Ak, B. Arnrich, G. Salur and C. Ersoy, "Inertial Sensor-Based Robust Gait Analysis in Non-Hospital Settings for Neurological Disorders", *Sensors*, Vol. 17, No. 4, 2017.

78. Kamouskos, S., "Supporting nomadic users within virtual private networks", *IEEE Service Portability and Virtual Customer Environments*, pp. 128–133, 2000.

79. Uehara, M., "Mist Computing: Linking Cloudlet to Fogs", R. Lee (Editor), *Computational Science/Intelligence and Applied Informatics*, pp. 201–213, Springer International Publishing, Cham, 2018.

80. Ceselli, A., M. Premoli and S. Secci, "Mobile Edge Cloud Network Design Optimization", *IEEE/ACM Transactions on Networking*, Vol. 25, No. 3, pp. 1818–1831, June 2017.

81. Tong, L., Y. Li and W. Gao, "A hierarchical edge cloud architecture for mobile computing", *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, April 2016.

82. Zhou, D., F. Chao, C. M. Lin, L. Yang, M. Shi and C. Zhou, "Integration of fuzzy CMAC and BELC networks for uncertain nonlinear system control", *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–6, July 2017.

83. Salmani, V., R. Ensafi, N. Khatib-Astaneh and M. Naghibzadeh, "A Fuzzy-Based Multi-criteria Scheduler for Uniform Multiprocessor Real-Time Systems", *10th International Conference on Information Technology (ICIT 2007)*, pp. 179–184, Dec 2007.

84. Tsuzaki, Y. and Y. Okabe, "Reactive configuration updating for Intent-Based Networking", *2017 International Conference on Information Networking (ICOIN)*, pp. 97–102, Jan 2017.

85. Google, *Google Cloud Platform*, 2011, `https://cloud.google.com`, accessed in May 2020.

86. Sabella, D., A. Vaillant, P. Kuure, U. Rauschenbach and F. Giust, "Mobile-Edge Computing Architecture: The role of MEC in the Internet of Things", *IEEE Consumer Electronics Magazine*, Vol. 5, No. 4, Oct 2016.

87. Imagane, K., K. Kanai, J. Katto, T. Tsuda and H. Nakazato, "Performance evaluations of multimedia service function chaining in edge clouds", *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–4, Jan 2018.

88. Ghosh, S., Q. Razouqi, H. J. Schumacher and A. Celmins, "A survey of recent advances in fuzzy logic in telecommunications networks and new challenges", *IEEE Transactions on Fuzzy Systems*, Vol. 6, No. 3, pp. 443–447, Aug 1998.

89. Abdullah, L., "Fuzzy Multi Criteria Decision Making and its Applications: A Brief Review of Category", *Procedia - Social and Behavioral Sciences*, Vol. 97, pp. 131 – 136, 2013.

90. Mendel, J. M., "Fuzzy Logic Systems for Engineering: A Tutorial", *Proceedings of the IEEE*, Vol. 83, pp. 345–377, March 1995.

91. Mokarram, M., A. Khoei, K. Hadidi and K. Gheysari, "Implementation of Centroid Defuzzifier Block Using CMOS Circuits", *4th International IEEE Conference on Intelligent Systems*, Vol. 1, pp. 2–29, September 2008.

92. Sonmez, C., A. Ozgovde and C. Ersoy, "Performance evaluation of single-tier and two-tier cloudlet assisted applications", *IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 302–307, May 2017.

93. Meneguette, R. I., A. Boukerche and A. H. M. Pimenta, "AVARAC: An Availability-Based Resource Allocation Scheme for Vehicular Cloud", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 20, No. 10, pp. 3688–3699, Oct 2019.

94. Arbib, J. and T. Seba, *Rethinking Transportation 2020-2030*, RethinkX Sector Disruption, RethinkX, 2017, `https://books.google.com.tr/books?id=lLMtswEACAAJ`.

95. Litman, T., *Autonomous Vehicle Implementation Predictions: Implications for Transport Planning*, desLibris: Documents collection, Victoria Transport Policy Institute, 2013, `https://books.google.com.tr/books?id=G3FKnwEACAAJ`.

96. Curry, E. and A. Sheth, "Next-Generation Smart Environments: From System of Systems to Data Ecosystems", *IEEE Intelligent Systems*, Vol. 33, No. 3, pp. 69–76, May 2018.

97. Atallah, R. F., C. M. Assi and M. J. Khabbaz, "Scheduling the Operation of a

Connected Vehicular Network Using Deep Reinforcement Learning", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 20, No. 5, pp. 1669–1682, May 2019.

98. Zhang, K., Y. Mao, S. Leng, S. Maharjan and Y. Zhang, "Optimal delay constrained offloading for vehicular edge computing networks", *IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2017.

99. Payalan, Y. F. and M. A. Guvensan, "Towards Next-Generation Vehicles Featuring the Vehicle Intelligence", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 21, No. 1, pp. 30–47, Jan 2020.

100. Grewe, D., M. Wagner, M. Arumaithurai, I. Psaras and D. Kutscher, "Information-Centric Mobile Edge Computing for Connected Vehicle Environments: Challenges and Research Directions", *Proceedings of the Workshop on Mobile Edge Communications*, MECOMM '17, pp. 7–12, ACM, New York, NY, USA, 2017.

101. Liu, S., L. Liu, J. Tang, B. Yu, Y. Wang and W. Shi, "Edge Computing for Autonomous Driving: Opportunities and Challenges", *Proceedings of the IEEE*, Vol. 107, No. 8, pp. 1697–1716, 2019.

102. Nobre, J. C., A. M. de Souza, D. Rosário, C. Both, L. A. Villas, E. Cerqueira, T. Braun and M. Gerla, "Vehicular software-defined networking and fog computing: Integration and design principles", *Ad Hoc Networks*, Vol. 82, pp. 172 – 181, 2019.

103. Tong, W., A. Hussain, W. X. Bo and S. Maharjan, "Artificial Intelligence for Vehicle-to-Everything: A Survey", *IEEE Access*, Vol. 7, pp. 10823–10843, 2019.

104. Wang, Y., S. Wang, S. Zhang and H. Cen, "An Edge-Assisted Data Distribution Method for Vehicular Network Services", *IEEE Access*, Vol. 7, pp. 147713–147720,

2019.

105. Zhan, W., C. Luo, J. Wang, C. Wang, G. Min, H. Duan and Q. Zhu, "Deep Reinforcement Learning-Based Offloading Scheduling for Vehicular Edge Computing", *IEEE Internet of Things Journal*, 2020.

106. Liu, G., Y. Xu, Z. He, Y. Rao, J. Xia and L. Fan, "Deep Learning-Based Channel Prediction for Edge Computing Networks Toward Intelligent Connected Vehicles", *IEEE Access*, Vol. 7, pp. 114487–114495, 2019.

107. Peng, H. and X. S. Shen, "Deep Reinforcement Learning based Resource Management for Multi-Access Edge Computing in Vehicular Networks", *IEEE Transactions on Network Science and Engineering*, 2020.

108. Sami, H., A. Mourad and W. El-Hajj, "Vehicular-OBUs-As-On-Demand-Fogs: Resource and Context Aware Deployment of Containerized Micro-Services", *IEEE/ACM Transactions on Networking*, Vol. 28, No. 2, pp. 778–790, 2020.

109. Peng, H., Q. Ye and X. Shen, "Spectrum Management for Multi-Access Edge Computing in Autonomous Vehicular Networks", *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, 2019.

110. John, G. H. and P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers", *Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 338–345, Morgan Kaufmann, San Mateo, 1995.

111. Platt, J., "Fast Training of Support Vector Machines using Sequential Minimal Optimization", B. Schoelkopf, C. Burges and A. Smola (Editors), *Advances in Kernel Methods - Support Vector Learning*, MIT Press, 1998.

112. Gardner, M. and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences", *Atmospheric Environment*, Vol. 32, No. 14, pp. 2627 – 2636, 1998.

113. Lin, W.-Y., M.-W. Li, K.-C. Lan and C.-H. Hsu, "A Comparison of 802.11a and 802.11p for V-to-I Communication: A Measurement Study", *Quality, Reliability, Security and Robustness in Heterogeneous Networks*, pp. 559–570, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

114. Xu, Z., X. Li, X. Zhao, M. Zhang and Z. Wang, "DSRC versus 4G-LTE for Connected Vehicle Applications: A Study on Field Experiments of Vehicular Communication Performance", *Journal of advanced transportation*, Vol. 435, 08 2017.

115. Sonmez, C., *EdgeCloudSim Google Discussion Forum*, 2018, `https://groups.google.com/forum/#!forum/edgecloudsim`, accessed in May 2020.

116. Sonmez, C., *EdgeCloudSim YouTube Channel*, 2018, `https://www.youtube.com/channel/UC2gnXTWHHN6h4bk1D5gpcIA`, accessed in May 2020.

117. Martínez, Y., A. Nowé, J. Suárez and R. Bello, "A Reinforcement Learning Approach for the Flexible Job Shop Scheduling Problem", *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, pp. 253–262, Springer-Verlag, Berlin, Heidelberg, 2011.