

FAIR SCHEDULING RESOURCE OPTIMIZATION IN SDN-BASED EDGE COMPUTING

Thesis Submitted for the Award of the Degree of

DOCTOR OF PHILOSOPHY

in

Computer Applications

By

Ajay Nain

Registration Number: 12105463

Supervised By

Dr. Sophiya Sheikh (26298)

Department of Computer Applications
(Associate Professor)

Lovely Professional University

Co-Supervised By

Dr. Mohammad Shahid

Department of Commerce
(Associate Professor)

Aligarh Muslim University,
Uttar Pradesh, India.



LOVELY PROFESSIONAL UNIVERSITY, PUNJAB

2026

DECLARATION

I, hereby declared that the presented work in the thesis entitled, “ **Fair Scheduling Resource Optimization in SDN-based Edge Computing**” in fulfillment of degree of Doctor of Philosophy (Ph.D.) is the outcome of research work carried out by me under the supervision of Dr. Sophiya Sheikh, working as Associate Professor in Computer Application, in the School of Computer Applications of Lovely Professional University, Punjab, India. In keeping with the general practice of reporting scientific observations, due acknowledgements have been made whenever the work described here has been based on the findings of another investigator. This work has not been submitted in part or full to any other University or Institute for the award of any degree

(Signature of Scholar)

Ajay Nain

Registration No.:12105463

School of Computer Applications,

Lovely Professional University, Punjab, India.

CERTIFICATE

This is to certify that the work reported in the Ph.D. thesis entitled, “**Fair Scheduling Resource Optimization in SDN-based Edge Computing**” submitted in fulfillment of the requirement for the reward of degree of Doctor of Philosophy (Ph.D.) in the School of Computer Applications of Lovely Professional University, Punjab, India, is a research work carried out by Ajay Nain, Registration No. 12105463, is Bonafide record of her original work carried out under my supervision and that no part of thesis has been submitted for any other degree, diploma or equivalent course

(Signature of Supervisor)

Dr. Sophiya Sheikh

Associate Professor

Department of Computer Applications,

School of Computer Application,

Lovely Professional University, Punjab, India

ACKNOWLEDGEMENT

This thesis marks the conclusion of a journey that has been both difficult and rewarding. I am genuinely thankful to everyone whose support, guidance, and encouragement have meant a great deal to me. First and foremost, I want to sincerely thank Lord Shiva, the Most Gracious and Merciful, for providing me with the strength, wisdom, and perseverance to complete this work.

I want to sincerely thank my Supervisor, **Dr. Sophiya Sheikh**, whose unwavering support, wisdom, and expertise have been the foundation of this entire research journey. From the very beginning, Dr. Sheikh, you have been more than just a guide; you have been a source of strength, encouragement, and emotional reassurance. Your thoughtful feedback, patience, and consistent motivation helped me through moments of uncertainty and self-doubt. I am truly grateful to have a supervisor who believed not only in the potential of my work but also in me as a person. Your compassion and dedication have made a significant impact on my academic and personal growth, and I will always cherish your role in this process. This journey wouldn't have been possible without your emotional and intellectual support, and for that, I am forever thankful.

I also want to sincerely thank my Co-Supervisor, **Dr. Mohammad Shahid**, from the Department of Commerce at Aligarh Muslim University, Aligarh, Uttar Pradesh, India. Your ongoing support, kind mentorship, and valuable insights have meant a lot to me. Your encouragement provided strength during tough times and greatly enhanced this thesis.

I sincerely thank **Dr. Manmohan Sharma**, **Dr. Avinash Bhagat**, and **Dr. Amar Singh** for their invaluable advice and support throughout my research. My appreciation also goes to **Mr. Ajay Kumar Bansal**, HOD of the School of Computer Applications at Lovely Professional University, Punjab. Their inspiring leadership has fostered a positive and encouraging environment for research and learning, which played a crucial role in the successful completion of this thesis. I am genuinely grateful for their steadfast support and the opportunities they've given me. Lastly, I want to warmly thank all the faculty members of the School of Computer Applications at Lovely Professional University, Punjab. To my RAC members, your constructive critiques and invaluable suggestions have significantly enriched this research. Your diverse perspectives and expertise have been instrumental in refining my ideas and pushing the boundaries of my work.

I am also deeply thankful to my colleagues and friends in Lovely Professional

University. Your camaraderie, collaborative spirit, and intellectual exchange have made this journey enjoyable, memorable, and intellectually stimulating. Special thanks to Dr. Rohit Malik, Pawan Bhaker, Dr. Pinki, Dr. Kapil Yadav, Dr. Sheikh Umar Mushtaq, Dr. Avtar Singh, Dr. Sandeep Sheokand, Dr. Rishi, Atul and Mohit Yadav. This research would not have been possible without their emotional support and invaluable help assistance.

A heartfelt thank you to my family, whose constant love and faith in me have truly been my foundation. To my wonderful parents, whose words and actions instilled in me the strength of perseverance and the importance of hard work, your patience, love, and unwavering support have truly been my anchor throughout this journey. Your unwavering belief in me, especially during times when I doubted myself, gave me the courage to keep going. I am deeply grateful for you being my rock every step of the way. To my amazing siblings, your constant encouragement and understanding mean the world to me. I am forever thankful for your sacrifices and endless support, which made this achievement possible. Thank you all from the bottom of my heart!

Lastly, I dedicate this thesis to my beloved family **Late Ch. Rohia Ram Nain** (Grandfather), **Sh. Sewa Singh** (Father), **Smt. Krishna Devi** (Mother), **Annu** and **Anju** (Sisters), **Amit** (Brother), **Suresh** and **Pawan Sheokand** (Maternal Uncles), whose inspiration continues to guide me. Your legacy of passion and dedication consistently reminds me of the values I aim to uphold. This journey has been a beautiful mosaic filled with countless moments of learning, perseverance, and growth. To everyone who has been part of this journey, thank you so much for your trust, support, and encouragement. Your presence has truly made this experience more meaningful and memorable.

This thesis truly belongs to all of us, embodying our shared efforts and collective spirit.

Abstract

The rapid expansion of networking and computing technologies emphasises the importance of maximizing resource efficiency and simplifying processes. In the current landscape, real-time applications demand quick data processing and effective management. Traditional computing systems can handle complex services, and conventional Cloud Computing provides ample storage and processing capacity for user tasks. However, cloud servers are located far from user devices; offloading tasks to the cloud leads to transmission delays, higher latency, and reduced QoE. To bridge the gap between user devices and cloud servers, a new technology called edge computing is introduced, deploying servers near the devices or at the edge of the network. Edge computing addresses the low latency and local data processing requirements of various users. It is a paradigm that decentralizes data processing and storage and brings these facilities closer to the source of data generation.

Edge computing is inherently distributed; therefore, managing edge devices and monitoring the entire network system requires a control system that can oversee the network, traffic, and facilitate the implementation of OpenFlow protocols. The challenges encountered by edge computing can be addressed by integrating a centralized controller, known as SDN. SDN enables dynamic resource allocation and flexible network provisioning by separating the control plane from the data plane and supporting centre-based network management. It also offers a comprehensive view of the network, aiding in efficient resource selection and workload adjustments. SDN utilizes protocols that enable the implementation of various protocols on a single software platform, with these protocols operating on hardware devices. It further helps to overcome the challenges faced by both distributed and centralised standards individually. Additionally, this integrated approach is referred to as SDN-based edge computing.

In SDN-based edge computing, resource allocation and load balancing face significant challenges due to the dynamic and diverse nature of edge environments. Typically, edge nodes operate with limited resources, requiring efficient strategies to prioritise critical tasks while maintaining fairness. Multiple edge nodes may have various workloads and different capabilities, such as varying levels of computing power, storage capacity, and network bandwidth. Furthermore, many edge

applications are latency-sensitive, such as real-time monitoring and autonomous systems, making resource allocation difficult because delays can significantly impact system performance. Network congestion and unpredictable traffic patterns worsen these issues by often causing tasks to be unevenly distributed, overloading some nodes while underutilising others. Balancing these factors is essential to prevent bottlenecks, optimise resource use, and sustain consistent performance across the edge network. Consequently, allocating resources and distributing the load effectively is complex.

This thesis studies the design and development of fair and efficient strategies for task scheduling, resource allocation, and load balancing within SDN-based edge computing environments. The proposed methods enable real-time monitoring of edge resources and adaptive allocation using centralized control and global visibility, enabled by SDN. The focus is on ensuring fair task distribution to prevent server overload, optimize resource utilization, and improve system performance. Moreover, to address the diverse and changing needs of edge computing, several models covering different combinations of scheduling, allocation, and load balancing have been created. The main goal of these contributions is to enhance performance and resource use while reducing makespan, latency, and total completion time in an SDN-based edge paradigm.

The thesis starts with an Introduction chapter, i.e., Chapter 1, that explains the motivation, problem statement, and objectives of the study. A detailed Literature Review follows in Chapter 2, which examines current scheduling methods in edge and SDN-based edge computing, focusing on various research optimization areas like resource allocation, task scheduling, computation offloading, and load balancing, while also identifying gaps in current studies. To improve resource optimization, Chapter 3 proposes a hybrid resource allocation strategy that combines PSO and GWO to reduce energy consumption and delay. Chapter 4 introduces ABRL, which optimizes task scheduling, resource allocation, and load balancing to optimize makespan, average resource utilization, and load balance. The performance of ABRL is compared with FCFS, SJF, DWECS, Max-Min, Min-Min, PTAL, and ACTE on both small and large task datasets, with task counts ranging from 100 to 750 and 1000 to 10000, across various numbers of servers. Additionally, Chapter 5 presents the DPTS model, which uses clustering to improve task-to-VM mapping by considering priority constraints. The scheduling process involves several steps, including transmission from user devices to the server and the VMs within the edge server. DPTS is evaluated by comparing OSO, UTO, and

FF at different levels of heterogeneity, such as low and high, within a task range of 100 to 500. The thesis concludes in Chapter 6 with key findings, contributions, and directions for future research. This work advances the development of edge computing frameworks capable of providing scalable, fair, and resource-efficient services in today’s distributed computing environments.

List of Abbreviations

QoE	Quality of Experience
SDN	Software Defined Network
QoS	Quality of Service
IoT	Internet of Things
RMC	Resource Monitoring and Control
NFV	Network Function Virtualization
SLAs	Service-Level Agreements
WRR	Weighted Round Robin
WFQ	Weighted Fair Queuing
FCFS	First Come First Serve
EDF	Earliest Deadline First
DWECS	Dynamic Weighted Edge Computing Scheduler
SJF	Shortest Job First
PTAL	Parallelization Task Allocation and Load Balancing Algorithm
ACTE	Assigning Computing Task Among Multiple Edge Servers
OSO	Optimal IoT Service Offloading
FF	First Fit
UTO	Utility Aware Task Offloading
PSO	Particle Swarm Optimization
GWO	Grey Wolf Optimization
SLR	Systematic Literature Review
MEC	Mobile Edge Computing
ES	Edge Servers
AR	Augmented Reality
P2P	Peer to Peer
VMs	Virtual Machines

NF	Network Function
M2M	Machine to Machine
MAEC	Multi Access Edge Computing
ML	Machine Learning
CC	Cloud Computing
AI	Artificial Intelligence
CoT	Cloud of Things
UE	User Equipments
ECs	Edge Cloudlets
FBSs	Femto Base Stations
EN	Edge Nodes
RC	Remote Cloud
ABRL	Average Based Resource Allocation and Load Balancing
API	Application Programming Interface
LLB	Level of Load Balancing
DPTS	Dynamic Priority Task Scheduling
MIPS	Million Instructions Per Second
MI	Million Instructions

List of Figures

1.1	Edge Computing Architecture	4
1.2	SDN Architecture	9
1.3	SDN-based Edge Computing Architecture	13
1.4	Flowchart of Thesis Organization	34
2.1	Percentage of Manuscripts from Frequent Sources	38
2.2	Number of Articles (Year Wise)	38
2.3	Flow Chart of Inclusion Exclusion Criteria	40
2.4	Schematic Diagram of Adopted Methodology	41
2.5	Optimized Parameters	69
3.1	Architecture of SDN-based Networks	74
3.2	Flow chart of Proposed Algorithm	78
3.3	Time Delay Comparison	80
3.4	Energy Optimization Comparison	81
4.1	Proposed Architecture for SDN-based Edge Computing	86
4.2	Flow Diagram of Resource Allocation to Edge Servers	96
4.3	Flow of Load Balancing And Migrated Task's	97
4.4	1. Makespan results on 20 servers for task sizes 100-750, 2. Makespan results on 20 servers for task size 1000-10000	102
4.5	1. Makespan results on 50 servers for task sizes 100-750, 2. Makespan results on 50 servers for task size 1000-10000	102

4.6	1. Average Resource Utilization results on 20 servers for task sizes 100-750, 2. Average Resource Utilization results on 20 servers for task size 1000-10000	103
4.7	1. Average Resource Utilization results on 50 servers for task sizes 100-750, 2. Average Resource Utilization results on 50 servers for task size 1000-10000	103
4.8	1. Level of load balancing results on 20 servers for task sizes 100-750, 2. Level of load balancing results on 20 servers for task size 1000-10000	104
4.9	1. Level of load balancing results on 50 servers for task sizes 100-750, 2. Level of load balancing results on 50 servers for task size 1000-10000	104
5.1	Proposed Architecture for SDN-based Edge Computing	113
5.2	Simulation Sample of tasks and server deployment	124
5.3	1. Average Resource Utilization for Varying Number of Tasks(Low Heterogeneity), 2. Average Resource Utilization for Varying Number of Tasks(High Heterogeneity)	126
5.4	1. Makespan for Varying Number of Tasks(Low Heterogeneity), 2. Makespan for Varying Number of Tasks(High Heterogeneity)	127
5.5	1. Latency for Varying Number of Tasks(Low Heterogeneity), 2. Latency for Varying Number of Tasks(High Heterogeneity)	128
5.6	1. Total Completion Time for Varying Number of Tasks(Low Heterogeneity), 2. Total Completion Time for Varying Number of Tasks(High Heterogeneity)	129
5.7	1. Level of Load Balancing for Varying Number of Tasks(Low Heterogeneity), 2. Level of Load Balancing for Varying Number of Tasks(High Heterogeneity)	130

List of Tables

2.1	Frequent Sources of Manuscript	37
2.2	Related Work on Resource Allocation and Scheduling	45
2.3	Related Work on Load Balancing and Distributions	49
2.4	Related Work on Computational Offloading and Rescheduling . .	53
2.5	Related Work on Optimization in SDN	58
2.6	A comparison of existing techniques in SDN-based edge computing	61
2.7	Summary of the Studies Proposed Algorithms in SDN-based Edge Computing	68
3.1	Simulation Parameters	80
4.1	Task size and execution time on ES_k	95
4.2	Configuration of task size and edge servers	98
4.3	Percent deviation of makespan for proposed approach from optimal value for 20 servers	99
4.4	Percent deviation of makespan for proposed approach from optimal value for 50 servers	100
4.5	Percent deviation of average resource utilization for proposed ap- proach from optimal value for 20 servers	100
4.6	Percent deviation of average resource utilization for proposed ap- proach from optimal value for 50 servers	100
4.7	Percent deviation of level of load balancing for proposed approach from optimal value for 20 servers	101

4.8	Percent deviation of level of load balancing for proposed approach from optimal value for 50 servers	101
5.1	Simulation Setup and Parameter	123

Contents

Abstract	ix
List of Abbreviations	xiii
List of Figures	xv
List of Tables	xvii
1 Introduction	1
1.1 History of Computing Technologies	2
1.2 Edge Computing	2
1.2.1 Advantages	5
1.2.2 Challenges	5
1.3 Need for Centralized Controller	7
1.4 Software-defined Networking	7
1.4.1 Advantages of SDN	9
1.4.2 Challenges of SDN	10
1.5 SDN-based Edge Computing	11
1.6 Resource Optimization	15
1.6.1 Need for Resource Optimization	15
1.6.2 Advantages of Resource Optimization	16
1.6.3 Challenges in Resource Optimization	17
1.7 Resource Optimization Aspects	18
1.7.1 Computation Offloading	18

1.7.2	Resource Allocation	20
1.7.3	Load Balancing	22
1.8	Fair Scheduling	24
1.8.1	Need for Fair Scheduling	25
1.8.2	Types of Fair Scheduling	25
1.8.3	Benefits of Fair Scheduling in SDN-based Edge Computing	27
1.8.4	Challenges of Fairness Scheduling in SDN-based Edge Computing	28
1.9	Fair Scheduling and Resource Optimization	29
1.10	Fair Scheduling in SDN-based Edge Computing	30
1.11	Motivation	31
1.12	Research Objectives	33
1.13	Thesis Organization	34
2	Review of Literature	35
2.1	Research Methodology and Plan of Study	36
2.1.1	Nature of Searching	37
2.1.2	Sources of Data	37
2.1.3	Duration and Validity of Study	38
2.1.4	Criteria for Inclusion and Exclusion	39
2.1.5	Review Judgement	40
2.1.6	Adopted Methodology	41
2.2	Related Work for Resource Optimizations Aspects in Edge Computing	42
2.2.1	Resource Allocation and Scheduling in Edge Computing .	43
2.2.2	Load Distribution and Balancing in Edge Computing . . .	48
2.2.3	Computational Offloading and Rescheduling in Edge Computing	51

2.3	Related Work for SDN Integration with Resource Optimization Aspects	55
2.3.1	Resource Optimization for SDN Integrated Paradigm . . .	55
2.3.2	Resource Optimization Techniques in SDN-based Edge Computing	60
2.3.3	SDN and Network Functions Virtualization (NFV)	63
2.3.4	Resource Provisioning at Edge Computing using SDN . . .	65
2.3.5	Dynamic Resource Management in Edge Computing using SDN	67
2.3.6	Resource Allocation in Multi-Access Edge Computing using SDN	67
2.3.7	Resource Management for Latency-sensitive Applications in Edge Computing using SDN	68
2.4	Open Issues, Challenges, and Current Trends for Integration of SDN-based Edge and IoT	69
2.5	Contextual Summary	72
3	Hybrid PSO-GWO for Resource Allocation in SDN-based Edge Computing	73
3.1	Proposed Model	73
3.1.1	System Architecture	74
3.1.2	Problem Formulation	75
3.1.3	Energy Formulation	76
3.1.4	The Algorithm	77
3.2	Experimental Setup and Result	79
3.3	Contextual Summary	82
4	An Average based Resource Allocation and Load Balancing Strategy in SDN-based Edge Computing	83
4.1	System Design	85

4.1.1	The System Architecture	85
4.1.2	Problem Formulation	88
4.1.3	The Proposed Algorithm	91
4.1.4	Time Complexity Analysis	94
4.1.5	Illustrative Example	94
4.2	Experimental Study	97
4.2.1	Experimental Setup	98
4.2.2	Results & Discussion	98
4.3	Contextual Summary	106
5	Dynamic Priority Task Scheduling for Resource Optimization in SDN-based Edge Computing	109
5.1	System Model	111
5.1.1	System Architecture	112
5.1.2	Problem Formulation	114
5.1.3	Problem-Defination	115
5.1.4	The Proposed Algorithm	120
5.1.5	Time Complexity Analysis of DPTS	121
5.2	Experimental Study	123
5.2.1	Experimental Setup	123
5.2.2	Results and Discussions	125
5.3	Contextual Summary	130
6	Conclusion and Future Advancements	133
6.1	Conclusion	134
6.1.1	Limitations	138
6.2	Future Work	138
	Bibliography	141

Chapter 1

Introduction

The rapid increase in Internet of Things (IoT) devices and applications has gained popularity over the past decades. Massive amounts of data are generated from these devices and sent to the cloud [1]. IoT devices typically have limited computation and storage capacity. Further, cloud computing allows [2] IoT devices to offload computation-intensive tasks and data to remote nodes. Data is processed in a cloud system by a set of linked servers. However, IoT devices may congest the network while data is transferred to the cloud. Moreover, the cloud [3-4] nodes may simultaneously be overwhelmed with tasks from heterogeneous sources. Due to the heavy load between the device and the cloud server, the execution of real-time applications on the IoT device may be shaky or considerably impaired.

The traditional cloud servers cannot handle the massive volume of data, necessitating the development of a more optimal management system for computing in real-time IoT applications. Besides, the cloud servers are deployed very far from the end users, which increases the latency and affects the system's overall performance. Therefore, there is a need for efficient computing technology to overcome the issues faced by traditional cloud computing. One of the possible solutions is bringing computing capabilities to the network's edge. A promising technology called edge computing has emerged [5]. This platform improves data processing by locating computing resources closer to data sources. As a result, it offers faster processing speeds, reduced network latency, and enhanced user experience.

1.1 History of Computing Technologies

The advancement of computing technology began with centralized systems and progressed toward more distributed forms to meet the increasing demands for data and processing. The need for the IoT is identified because of the rapid advancement of infrastructure, including sensors, embedded systems, and the internet. The term IoT was introduced by Kevin Ashton in 1999. It was further accelerated in 2006 with the emergence of cloud computing by Amazon and its peers, offering scalable resources and services to billions of devices. However, the rapid growth of IoT brought problems like slower response times, limited bandwidth, and the need for real-time data processing, showing the weaknesses of centralized cloud models. These limitations led to the emergence of edge computing as a solution. Further, this led to the development of edge computing in the early 2010s, which aimed to reduce latency and bandwidth consumption by processing data closer to the source. The emergence of 5G technology integrated with intelligence systems made edge computing pivotal for time-sensitive applications, smart city frameworks, autonomous technologies, intelligent industrial automation, and beyond.

1.2 Edge Computing

Over the last few years, there has been a significant increase in global interconnectivity, resulting in a corresponding surge in the generation and transmission of data across networks. Edge Computing [6-7] was introduced as a solution to bring computing technologies closer to data generation devices as an alternative to depending on centralized data centers or clouds. Edge computing aims to minimize latency, increase efficiency, and improve the reliability of computer systems. The key point for Edge Computing is its ability to process and analyze data closer to its source. Edge computing devices are deployed near the physical devices or at the network edge. Edge servers are positioned to reduce the data volume that must be sent to distant cloud data centers. This, in turn, enables dynamic decision-making and faster response times, making it a perfect solution for run-time applications, i.e., IoT, autonomous vehicles, industrial automation, smart cities, etc. [8-9]. Edge computing can increase productivity in industrial automation by processing data in place, in real time. It can support decision-making in autonomous vehicles by processing information from sensors quickly.

Also, by enabling local decision-making in IoT devices, Edge computing lessens the need for constant connectivity with the internet and speeds up response time. Besides, due to the tremendous use of these technologies, the data created and produced from devices and sensors at the network edge continues to grow, and edge computing is becoming increasingly important.

Imagine a smart city where IoT devices are installed to monitor traffic, detect air pollution levels, and manage waste collection. With Edge Computing, these devices can communicate with each other and process data locally. For example, traffic sensors can communicate with streetlights to optimize [10] traffic flow, reducing congestion and carbon emissions. Waste collection trucks can be routed more efficiently based on real-time data on waste levels in different parts of the city. This approach reduces costs, improves efficiency, and enhances the quality of life for residents.

Processing/computing servers are located at the network's periphery in the edge computing architecture to carry out calculations and offer storage services within the scope of a radio access network. Enhancing the end user's Quality of Experience (QoE) is the primary goal of edge architecture, achieved by minimizing response time, latency, and throughput. It enables instant decision-making for a variety of time-sensitive services and real-time applications.

Since Edge computing possesses various applications and advantages over cloud computing, it poses some challenges when hosting on the network. To analyze the multiple benefits and challenges of edge computing, it is essential to study the architecture of Edge Computing in the context of what it does and how it can be implemented. Unlike cloud computing, edge computing embeds computational edge nodes near the network, also known as edge servers, and acts as mini-cloud data centres. Generally, the Edge Computing architecture [11] is divided into three aspects: the front end specifies the data generation layer or IoT devices, the near end defines edge nodes, and the far end comprises Cloud servers [8], as shown in Figure 1.1, discussed below.

Front End/ Physical Layer/ Device Layer: The end devices (e.g., IoTs, sensors) are strategically positioned at the forefront of the edge computing architecture. The frontend environment has the potential to enhance user interaction and improve responsiveness for end users. Edge computing can provide real-time services for some applications, leveraging the abundant computing capacity of various nearby end devices. Since these end devices possess limited capabilities,

most objectives cannot be adequately fulfilled inside the end device environment [12]. Therefore, the end devices must transmit on-demand resources to the servers in these instances.

Near End/ Middle Layer/ Edge Layer: The gateways implemented in the middle layer near the device layer will facilitate most of the network traffic streams. Edge servers may have various resource requirements for real-time data processing and reduction, data caching and buffering, compute offloading, and response control. In edge computing, a significant portion of data computation and storage is transferred to edge servers near the end devices. As a result, the end devices can enhance their performance in data computing and storage, with a slight decrease in latency.

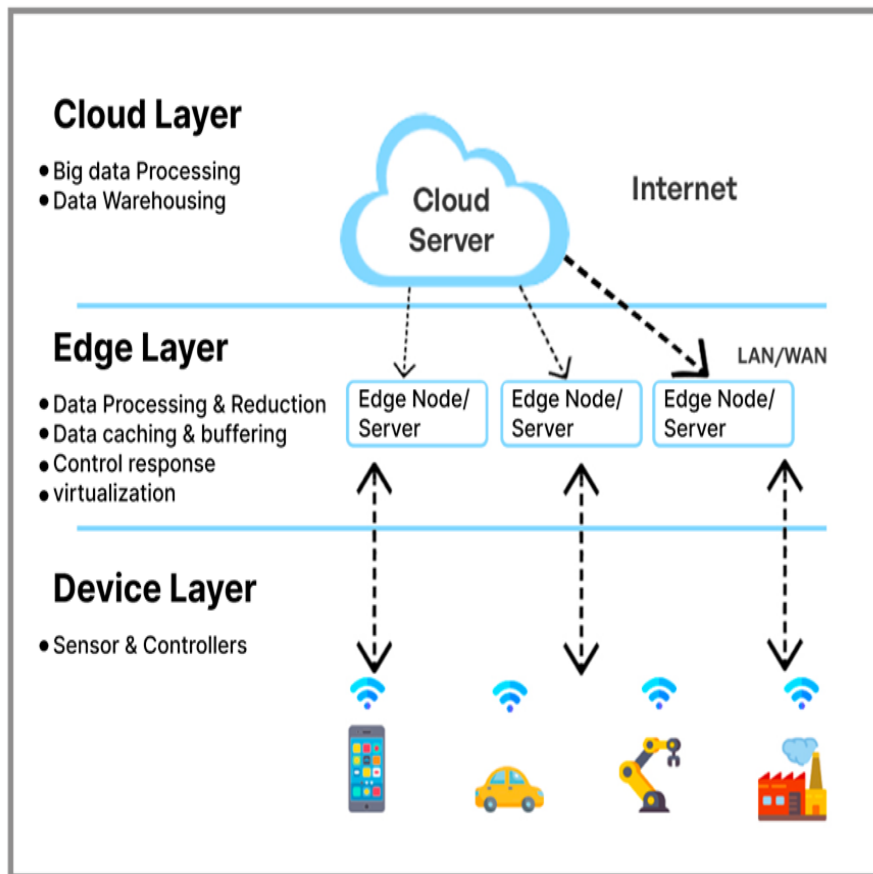


Figure 1.1: Edge Computing Architecture

Far End/Cloud Layer: When cloud servers are placed away from data-creating devices, there are noticeable delays in data transmission across the networks. However, cloud servers located in remote environments can provide increased computing power and data storage capabilities. Cloud servers offer many solutions in

massively parallel data processing, big data management and mining, and machine learning [13-14].

1.2.1 Advantages

The architecture of edge computing has several advantages over a traditional network, as discussed below:

- **Low Latency:** Edge computing is a technology that significantly reduces the time it takes for data to travel from where it is generated to where it is processed. This helps to minimize any delays [15] that may occur during this process. This has many benefits in different applications, particularly in tasks that need immediate responses, as it ensures faster decision-making due to being deployed at the very edge.
- **Bandwidth Optimization:** Edge devices can process data locally, handling various data processing tasks without sending them to a central server or cloud. This helps minimize the time of data transmission and improves overall efficiency. Conserving network bandwidth [16] makes the system more efficient and cost-effective and helps reduce traffic jams in the network.
- **Real-Time Analytics:** Edge devices can analyze incoming data in real time [17], generating instant insights. This technology is crucial when making run-time decisions, such as predicting maintenance needs in industrial settings.
- **Scalability:** Edge computing can adjust its capacity [18] to handle diverse workloads and the number of devices involved. Ensuring scalability is crucial for applications that adopt changes on demand for the number of connected devices. It also helps in efficiently allocating resources.

1.2.2 Challenges

Edge computing architecture faces several challenges identified in the existing literature review, which are discussed below.

- **Computation Offloading:** It is the process of migrating tasks from edge devices to remote cloud devices. When tasks are to be executed on the

edge server, selecting the appropriate edge server and remote cloud server for processing can be challenging. A better offloading [19] scheme helps distribute the load adequately across all edge servers, making the system highly efficient.

- **Privacy and Security:** This challenge is significant due to the distributed and resource-constrained nature of the edge environment. The edge servers are deployed near the data-generating devices; therefore, data privacy [20], prevention of data leakage, and unauthorised access are significant challenges in the edge environment. Besides, edge devices are generally deployed in open-access areas. Hence, the security of network devices during data transmission from edge devices to another server is also a significant concern [21]. Moreover, attackers accessing edge servers can potentially steal or alter the information.
- **Resource Management:** In Edge computing, devices and networks are connected to produce/execute heterogeneous data that follows various protocols. Managing these resources [22] is a crucial challenge in an edge environment. Therefore, networks and devices must be adequately managed to enhance device and network utilization.
- **Tasks Allocation between the Cloud and Edge:** The assignment of tasks between the cloud and the edge is a challenge because of the nature of the edge environment [23]. Some tasks are processed locally, and some are transmitted to the cloud. Therefore, efficient task scheduling approaches are needed to make the allocation more reliable and beneficial.
- **Energy Management:** The edge paradigm is distributed in nature. The edge executes several tasks simultaneously in real-time scenarios, making energy consumption substantial. Moreover, new energy protocols are required to improve the system's efficiency [24]. Several studies in edge computing focusing on energy harvesting and wireless charging capabilities [25-26]. Besides, resource scheduling becomes increasingly challenging when additional energy supplements are incorporated, as it requires the consideration of energy consumption during task transmission, task processing, and the utilisation of harvested energy.

1.3 Need for Centralized Controller

Edge computing involves heterogeneous devices spread across various locations, generating and processing data at the network's edge while handling communication and computation. This complexity poses significant challenges for overall control within the environment. It is a dynamic landscape where mobile users and real-time applications demand constant adaptation to current conditions. A centralized controller is necessary to provide a comprehensive view of the entire edge infrastructure, aiding in efficiently placing tasks, directing data, and managing bandwidth in ways that local nodes cannot achieve effectively. As devices differ in capability and function on various platforms, centralized management simplifies configuration, policy enforcement, and service deployment across the system. Additionally, real-time conditions can lead to rapid workload fluctuations, making dynamic resource allocation and load balancing vital and best coordinated through a centralized entity. This process entails effective resource management and allocation to enhance resource utilization and overall system performance.

To ensure smooth communication and cooperation between edge nodes, their status and operations need to be kept in sync, which a centralized controller can maintain efficiently and improve the system's resource management. To support such needs, a centralized controller like SDN plays a vital role by decoupling the control plane from the data plane, allowing the centralized controller to programmatically manage traffic flows, monitor resources, and enforce policies across all network elements. SDN enhances edge environments' flexibility, scalability, and intelligence, making it an ideal foundation for centralized control in the distributed edge computing paradigm.

1.4 Software-defined Networking

With the growing use and demand for technology, old technological concepts are becoming obsolete. A robust internet infrastructure is needed to cope with the increasing volume of data, especially given the rapid growth of IoT devices. SDN is paving the way for new network designs that meet the growing demands of users and network services. When network service demand changes frequently, service providers may use a shared network design to access necessary capabilities. The SDN controller can respond to fluctuating demands by collaborating with devices, applications, and routers. This also allows efficient resource utilization, flexible

configuration, and improved network performance [27].

SDN [28] has become one of the most significant network architectures due to scalability, flexibility, and resource monitoring. SDN improves resource utilization and information exchange, enabling applications to monitor network services and traffic. SDN is an innovative network management standard that separates the packet processing data plane from the routing processing control plane, creating a dynamic and coordinated system. In SDN, the control plane works like a centralized device known as a controller [29-30]. SDN helps management by decoupling the network devices' control and data planes. The controller is an advanced method for managing and optimizing network resources. It involves separating the power and data planes responsible for making network decisions and handling the actual data traffic. This further helps to ensure adequate resource allocation and better load balancing [31].

SDN enables the appropriate distribution and redistribution of network resources according to current demands and traffic patterns, which helps to optimize resource utilization. It can maximize resource usage by dynamically adjusting bandwidth, routing, and network policies through a central controller. SDN is a critical tool in modern network management because it offers flexibility that improves Quality of Service (QoS), enhances network performance, reduces operational costs, and optimizes resource allocation [32]. SDN architecture [33] comprises the Application Layer, Control Layer, and Infrastructure Layer. Figure 1.2 presents the layers of the SDN Architecture.

Application Layer: This layer is the open area in SDN where innovative applications can be implemented. Some examples of applications within this layer include Network Automation and Configuration. This layer is responsible for monitoring various systems, troubleshooting any issues, and ensuring QoS.

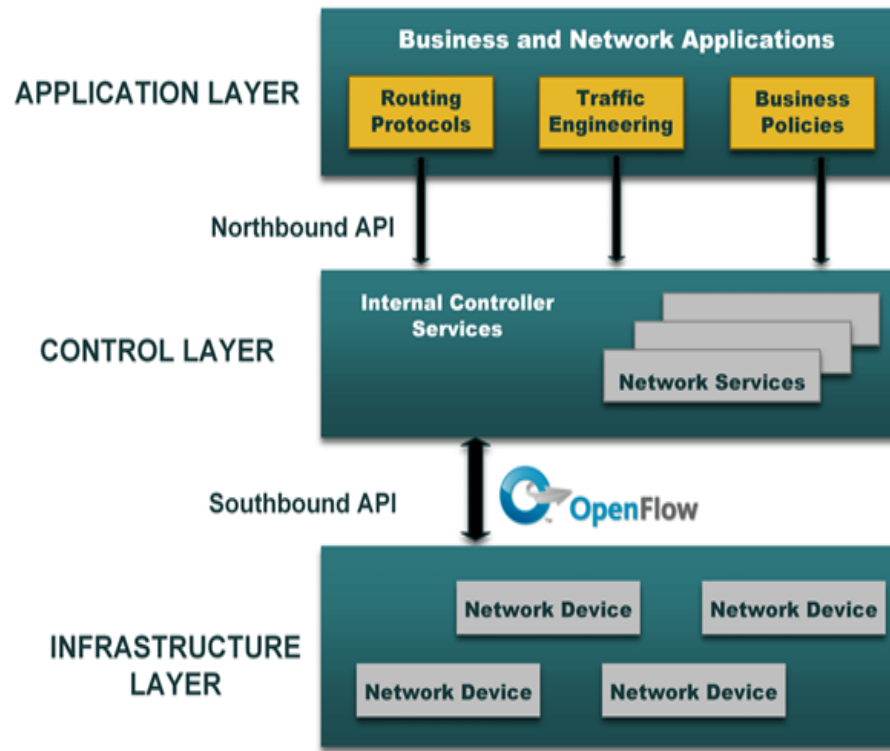


Figure 1.2: SDN Architecture

Control Layer: The control layer is the intelligent layer that contains the control plane logic. The intermediate layer is responsible for integrating the application and infrastructure layers. It is also used for monitoring business logic to gather and manage different types of network information, statistics, and details about the network's structure. This layer connects with the application layer through a Northbound interface and connects with the infrastructure layer using a Southbound interface, using the OpenFlow Protocol.

Infrastructure Layer: The infrastructure layer consists of various network devices that create the data plane and route traffic based on instructions received from the controller. Inside the data centre, there are operational network switches. SDN offers numerous advantages in the network industry, assisting organizations with better solutions [34]. These benefits include cost optimization, simplified troubleshooting, and efficient resource monitoring. Here are a few advantages of SDN.

1.4.1 Advantages of SDN

SDN brings various benefits to the network industry, like reducing operating costs by promoting virtualization in network environments. It allows for better resource

utilization, ultimately lowering operating expenses. Besides, it provides automation to networks to make them more reliable and customizable than ever [35]. Moreover, SDN-based networks can be further enhanced with automation. The following are the benefits of utilizing SDN.

- **Centralized Control:** SDN employs a central controller supervising the whole network. This method simplifies monitoring, handling network activities, and decision-making from a single location.
- **Flexible and Quick to Adjust:** SDN can quickly adjust network settings based on current needs. This function is proper when the user demands or data traffic changes.
- **Better Use of Resources:** Since the control and data functions are separate, SDN can use network devices more efficiently and reduce waste.
- **Easier Network Management:** In SDN, setting up devices and managing traffic can be automated, saving time and reducing mistakes.
- **Scalability:** In SDN, adding more devices or services to the network is easy, which benefits large or expanding systems.
- **Cost-Effective:** SDN allows communication between devices that follow different network protocols. It also reduces the cost of protocol modification and lowers costs through simpler operations.
- **Quality of Service:** SDN provides more control over data flow, allowing users to prioritize important tasks and improve performance [36].

1.4.2 Challenges of SDN

SDN introduces benefits such as centralized control, simplified network management, and improved flexibility. However, several researchers have observed various limitations/challenges specific to the existing literature. These limitations become more prominent in large-scale and distributed systems, particularly in environments that demand low latency, high availability, and efficient resource utilization. Identifying and addressing these challenges is essential for enhancing the overall performance and reliability of SDN-based networks:

- **Scalability Constraints:** In large-scale and distributed environments, centralized control in SDN may lead to scalability limitations. As the number of connected devices and data flows increases, the controller may struggle to process the growing volume of control requests efficiently. This may affect resource utilization and overall system performance.
- **Controller Bottleneck and Single Point of Failure:** When the system relies on the centralized controller, any failure at that point can affect the entire network, making it a critical vulnerability. This centralization can also slow down operations if the controller becomes overloaded. The request must be managed appropriately to prevent the controller from overloading.
- **Security:** One of the crucial challenges in SDN is security. Since SDN is centralized, all policy-related tasks are performed from the control layer or controller. The entire network can be compromised if someone gains unauthorized access to the controller. The DDoS attack is widely performed in SDN [37]. The current education systems and certification syllabi of network vendors like Cisco and Juniper do not provide enough information about SDN. This lack of knowledge and training on SDN is causing a shortage of skilled workforce. This will further cause many problems in the understanding of SDN architecture.

1.5 SDN-based Edge Computing

SDN is a novel approach that efficiently manages and controls resources and devices in edge computing environments [38]. SDN enables centralized network management by separating the control plane from the data forwarding plane, allowing greater flexibility and optimization in network operation processes. On the other hand, edge computing is a distributed paradigm involving data processing and storage being moved near its data source to reduce latency and improve real-time decision-making.

The integration of edge computing with SDN is a robust and rare combination of centralized and distributed architectural models that synergistically utilizes the advantages of both technologies. In this framework, SDN controllers supervise the network comprising edge devices and nodes, allowing them to have a global view of the network and efficiently allocate computing and communication resources in dynamic systems. This central control guarantees that the

network routes and the data exchange involving IoT endpoints, edge infrastructure, and cloud services are continuously optimized regarding performance and resource utilization [39].

The integration of these technologies makes a centralized and distributed paradigm [40]. It also helps to overcome the challenges faced by distributed and centralized standards individually. The emergence of SDN in edge computing enhances the performance and efficiency of the system. It can help to distribute the load proficiently among multiple edge servers, which further plays a vital role in resource allocation, task scheduling, and network monitoring [41-42]. The SDN follows the OpenFlow protocol that helps to implement the algorithm and many more protocols on just a single software, and they are operational on hardware devices.

Additionally, SDN-based Edge Computing is an innovative approach to managing and controlling IoT devices that enables organizations to leverage the benefits of both SDN and edge computing by providing a more efficient and secure network infrastructure. SDN can help organizations realize the full potential of IoT applications. SDN-based edge computing offers quicker data processing, increased dependability, and decreased latency, crucial for real-time IoT applications like continuous surveillance, automation, and control. This system enables businesses to use network resources more effectively and enhance the network infrastructure at a low cost [39].

In the SDN-based edge computing, SDN could apply advanced network policies, resource management, and security controls [43]. By centralizing network management, SDN enables organizations to enforce consistent policies across the network, reducing the risk of security breaches and improving overall network security.

Besides, SDN-based edge computing also faces some challenges, such as the need for interoperability, scalability, and the management of distributed resources that help to improve resource utilization. Ensuring that IoT devices can communicate with the SDN controller and that data is transferred securely and efficiently to the edge servers is also a critical concern in the SDN-based edge computing environments. Furthermore, the SDN-based edge computing Architecture is divided into different layers as shown in Figure 1.3 and discussed as follows:

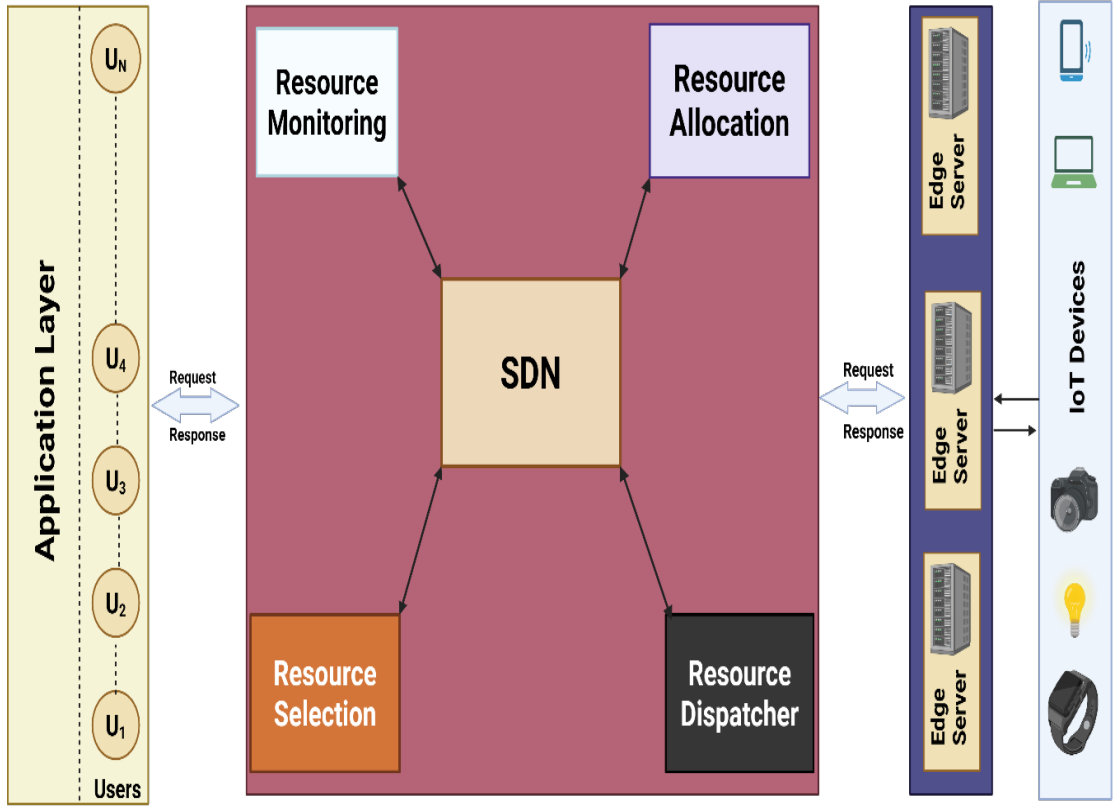


Figure 1.3: SDN-based Edge Computing Architecture

Application Layer: This is the topmost layer in the SDN-based edge computing architecture and is the primary interface between users and the edge computing environment. In this layer, multiple users generate and submit various task execution requests. These requests are then transmitted to the Control Layer for further processing, including task management, resource allocation, and scheduling. The Application Layer is a crucial gateway, enabling seamless user interaction with the system. It lies on the northbound side of the SDN layer.

Control Layer: This is the central layer of the system, acting as a bridge between the other two layers through northbound and southbound APIs. It is a middleware that manages communication among users, tasks, and servers. The SDN controller acts as the brain of the system. The controller oversees and allocates resources intelligently to ensure efficient task execution, which enhances the QoS and maintains overall system performance. It enables flexible resource management and effective load balancing. This layer also handles data coordination, system flow control, and seamless interaction between all architecture components. The components of the SDN layer are discussed below.

- (i) Resource Monitoring and Control: RMC stands for Resource Monitoring and Control, a framework for controlling, monitoring, and manipulating resources such as physical or logical system elements. The Hardware Management Console receives service events via RMC, which is used as a communication channel.
- (ii) Resource selection: The functions require data about the location of the species in issue and data on the resources accessible across the study region. Environmental and geographical variables, including categorical variables like land cover type and continuous variables like average rainfall over a specific period, can be used as resources.
- (iii) Resource allocation: Resource allocation is the practice of assigning and managing assets to support an organization's strategic goals. It involves managing tangible assets like hardware to maximize the use of softer assets like human capital.
- (iv) Resource dispatcher: A dispatcher is responsible for receiving emergency and non-emergency calls, checking driver logs, keeping records, managing problems, and dispatching suitable team members. They may also be responsible for monitoring weather reports and, if necessary, contacting authorities.

Infrastructure Layer: This layer is present southbound in the SDN control layer. It combines the edge, data forwarding, physical, and data creation layers. The primary responsibility of the infrastructure layer is to manage the physical and logical resources essential for the edge computing environment. It involves a variety of objects, sensors, and systems that capture and process data at the periphery, as well as network infrastructure for connecting these devices to the wider computational environment.

The edge layer plays a vital role in processing data locally by placing data sources near the network's edge. It helps to make communication faster and reduce latency. The data is generated from the physical layer to give the response to the user's request from the application layer. The edge servers are deployed on the edge layer to execute the task and are directly controlled by a centralized SDN controller. The control layer of the system directly controls the infrastructure layer. The aim is to design this environment to enhance resource utilization in the edge environment.

1.6 Resource Optimization

Resource optimization involves effectively managing and allocating different resources such as computing power, storage, bandwidth, energy, and time to improve performance while lowering operational costs and waste[44]. Resource monitoring, identification, demand forecasting, workload distribution, and automation are crucial to efficient resource utilization. Resource optimization aims to balance system performance, cost-effectiveness, scalability, and sustainability, often using advanced techniques like load balancing, task scheduling, auto-scaling, and machine learning-based prediction. It is crucial in achieving high efficiency, reliability, and responsiveness in cloud computing, edge computing, software-defined networking, or business operations.

1.6.1 Need for Resource Optimization

The demand for resource optimization arises from the increasing complexity of modern computing environments. As technologies like cloud computing, edge computing, and SDN evolve, there is a growing need to manage limited resources intelligently. The primary reasons include:

- **Systems with limited computing and communication resources:** Some systems contain limited processing power, memory, or data transfer capabilities. Effectively managing these limited resources is essential for the overall system's performance.
- **High user demand and dynamic workloads:** When many users access the system simultaneously or the workload is dynamic, resources must be allocated at run-time to keep the system's performance steady.
- **Increased costs and reduced system performance:** Inadequate resource allocation may lead to higher expenses and slower system responses, affecting overall efficiency.
- **For improved system stability, speed, and energy use:** Utilizing resources wisely helps the system run more smoothly, respond faster, and use less power, which is especially useful in energy-sensitive environments.
- **To provide consistent and dependable service:** Appropriate resource management ensures that all users receive reliable and uninterrupted service,

essential for users' trust and satisfaction.

1.6.2 Advantages of Resource Optimization

Resource optimization is essential for optimizing limited computing and communication resources. Adequate resource allocation, especially to the most crucial regions, enhances the performance of systems [45]. Resulting in enhanced efficiency, increased speed, and reduced operational costs. Some advantages of Resource Optimization are discussed below:

- **Efficient Utilization:** Optimizing resources proficiently ensures they are used to their maximum capacity without being overworked or unused. This leads to better system productivity and avoids unnecessary delays or downtime.
- **Cost Savings:** Optimizing resources helps lower overall expenses by reducing unnecessary hardware and software use. This reduces operational and infrastructure costs, leading to a more budget-friendly system.
- **Performance Enhancement:** System processes run faster and more efficiently when resources are managed effectively. This results in better response times, increased data handling ability, and improved user experience.
- **Load Balancing:** Resource optimization helps evenly distribute tasks across available systems or servers. This prevents resources from overloading and underloading, reducing the risk of crashes and improving system stability.
- **Energy Efficiency:** Enhanced resource optimization reduces energy consumption, which benefits both cost and environmental impact. Minimizing power usage also promotes sustainable computing.
- **Scalability:** Optimized systems can easily adjust to changing user demands. This means services can grow or shrink as needed without affecting performance or requiring major upgrades.
- **Reliability:** Effective resource utilization increases the system's ability to function without interruptions/disruptions. It lowers the chances of failure and ensures that services are consistently available to users.

1.6.3 Challenges in Resource Optimization

Resource optimization faces several challenges despite its benefits, especially in large-scale and real-time systems[46]. These are discussed below:

- **Resource Monitoring:** Monitoring the usage and status of distributed resources is essential to improving the system’s performance. However, consistently tracking these elements in real time, especially across different locations and platforms, poses a considerable challenge due to the complexity involved.
- **Dynamic Environments:** Workload levels and resource availability often shift without warning in distributed computing environments. These frequent changes make relying on static optimization methods complex and demand more responsive and adaptive strategies.
- **Complex Decision-Making:** Effective resource allocation must consider several QoS parameters, such as execution time, operational cost, and energy usage. Balancing these priorities increases the complexity of decision-making and requires well-designed algorithms to handle the trade-offs.
- **Scalability Issues:** As distributed systems expand in size and scope, managing and optimizing resources becomes increasingly complex. Larger systems introduce more variables and dependencies, complicating the optimization process unless scalable techniques are implemented.
- **Algorithm Overhead:** Some optimization processes require significant computing effort, which can add to the system’s workload. This added overhead may reduce system efficiency and offset the intended improvements in the considered environment with limited resources.
- **Heterogeneity:** Modern distributed systems are rarely uniform; they often incorporate various devices, platforms, and network configurations. This diversity necessitates optimization techniques that are flexible enough to function well in a heterogeneous environment.
- **Integration with Legacy Systems:** Many organizations still depend on outdated technologies that do not readily support recent optimization methods. Integrating such legacy systems often requires custom adjustments and may limit the full potential of modern optimization solutions.

1.7 Resource Optimization Aspects

Resource optimization involves efficiently allocating and utilizing network resources to ensure critical applications and services receive sufficient bandwidth to operate at peak efficiency [47]. This involves dynamic resource allocation based on real-time network requirements to achieve the most efficient network investments.

Considering the challenges of edge computing architectures, resource optimization plays a crucial role in maintaining performance and reliability. Technologies like SDN, NFV, network controllers, and various networking algorithms have been developed to address these needs. These innovations simplify network management, enhance flexibility, and reduce operational costs. They also support organizations in making better use of their network resources. As a result, resource optimization becomes vital for improving the productivity and efficiency of modern computer networks [48]. The various optimization methods are given below:

1.7.1 Computation Offloading

A distributed computing technique where computation tasks (generated from IoT devices) are offloaded to more powerful remote servers deployed at the network's edge or in the cloud [49]. This reduces the load on local computing devices and improves performance. The main goal of computation offloading is to enhance the system performance and reduce the latency for real-time application scenarios. The offloading decision can be static, dynamic, or modified according to the resource availability, network conditions, available bandwidth, and task complexity.

Edge computing leverages a wide range of computing devices at the network's periphery, including mobile phones, IoT devices, and network routers, to execute computational operations. On the other hand, cloud computing utilizes remote servers located in data centers to perform computational tasks. The computation offloading is categorized into several types based on what, when, where, and how the computation is offloaded [50]. Every type has a different purpose depending on the system's goals, such as reducing latency, response time, and enhancing the system's performance. The categorization based on various aspects is discussed below:

I. Based on the Scope of Offloading

Based on the scope, there are two types of offloading, which are discussed below.

- **Full Offloading:** This process occurs when the entire application or set of tasks is offloaded from the local IoT devices to a remote server in cloud or edge computing. This suits highly resource-intensive functions that exceed the local device's capabilities. It helps to improve processing efficiency and reduce the system's energy consumption; however, it is highly dependent on network stability.
- **Partial Offloading:** This process occurs when specific components or limited tasks are offloaded while the rest are executed locally. This type of offloading is used in applications based on modular architectures. It helps to create a balance between performance and communication overhead. It requires precise decision-making on the offloaded components to the remote server.

II. Based on Offloading Destination

There are two types of offloading based on the destination, which are discussed below.

- **Edge Offloading:** In this offloading mechanism, tasks are offloaded to the nearby edge servers or edge nodes located near the data sources or at the network's edge. Processing the data locally helps reduce latency and network congestion and helps decision-making for real-time application scenarios. However, the edge resources may have limited computation capabilities and cannot process heterogeneous data.
- **Cloud Offloading:** In cloud offloading, the tasks are sent to the centralized or traditional cloud servers with high computation capabilities. This offloading mechanism suits complex and heavy computations like deep learning methods. It allows access to a highly scalable and robust computational infrastructure that handles several tasks simultaneously. However, due to the deployment of cloud servers that are very far from data sources, latency and dependency on internet connectivity increase. It is not that effective in real-time application scenarios.

III. Based on Offloading Decisions

There are two types of offloading based on decision-making, as discussed below.

- **Static Offloading:** When the offloading decisions are made in advance, it is called static offloading. This offloading is useful when the workload is predictable. This is the more straightforward implementation and has a very low runtime overhead. However, it cannot be utilized with real-time changes in the environment or workload.
- **Dynamic Offloading:** The decisions are made at runtime based on the current resource availability, network conditions, and the task requirements in the dynamic offloading. Moreover, the system can change its decision according to the requirement and supports real-time application scenarios. Though it is a highly scalable, adaptive, resource-efficient mechanism, it still requires a lot of monitoring and dynamic algorithms to make further decisions.

1.7.2 Resource Allocation

This is an essential aspect of resource optimization that refers to the distribution of the available resources, such as CPU, memory, bandwidth, etc., to the different tasks/users in a manner that improves the system's resource utilization and performance [50]. Adequate resource utilization optimizes performance and reduces the latency of edge computing. Distinct resource allocation categories help achieve the proper resource distribution, as discussed below.

- I. **Static Resource Allocation:** This allocation strategy allocates the edge servers/resources to tasks/users at the initialization stage and is fixed throughout the execution process. It is simple to implement and enhances the performance of systems with predictable workloads. It does not adapt to changes in the process according to the workload and system conditions, making it more suitable for applications with fixed resource demands.
- II. **Dynamic Resource Allocation:** In this allocation strategy, the edge servers/ resources are allocated to tasks/users dynamically based on real-time conditions such as user demands, resource availability, and server load. These strategies are more flexible and responsive in case of uneven load. The system improves its performance and utilization by managing the allocation strategy according to user demand. In the edge computing scenario, these

strategies are more complex in implementation due to the regular need for monitoring and decision-making while working with heterogeneous environments.

III. Centralized Resource Allocation: A central authority, such as a controller (i.e., SDN) or scheduler, manages the allocation strategy to various tasks by continuously analysing the conditions. It is a supervised strategy using the dynamic approach. It overcomes the limitations of dynamic resource allocation strategies by having the capabilities of real-time monitoring and decision-making. It makes the allocation process straightforward by providing information on all the considerable aspects of the edge computing environment. When the centralized controller integrates with the distributed edge computing environment, it helps improve the system's performance through decision-making and monitoring.

IV. Distributed Resource Allocation: Resources are allocated decentralized, with multiple entities or nodes participating in the allocation process, or the servers/nodes are available for computation. This reduces the single point of failure, or if a single point has any error, it does not affect others and can handle large-scale effectively. The deployment of resources may be uneven, which makes coordination and communication between distributed edge servers/nodes difficult. It is widely used in large-scale distributed systems like edge computing and peer-to-peer networks.

V. Fair Resource Allocation: In this resource allocation strategy, the resources are allocated based on fairness principles, which ensure that all users/tasks receive an equal share of the resources. This promotes fairness by ensuring that all users get the needed share of the resources. In some scenarios, it may affect the system's performance for high-priority tasks and the user in real-time applications or emergencies.

VI. Priority-based Resource Allocation: The resources are allocated based on the priority levels assigned to different tasks/users, with high-priority tasks receiving resources first. It ensures that critical or high-priority tasks/users get the resources at the time they need for timely execution. This low-priority task may face delays or starvation if high-priority tasks are continuously arriving. It is mainly used in real-time systems, emergency response systems, and networks where specific tasks must be prioritized.

1.7.3 Load Balancing

Load balancing involves an adequate distribution of workloads across numerous computing devices. This technique ensures that no device is overburdened and that computing resources are utilized efficiently. In SDN-based edge computing, load balancing is essential to manage changing and diverse scenarios where tasks need quick responses [51]. SDN allows for a central control point and a complete network view, which helps make wise choices about where to send tasks based on current network status, server usage, applications need, etc. In edge computing, load balancing can occur at multiple levels, such as the network layer (e.g., traffic management and flow routing), the application layer (e.g., task scheduling and service placement), and the infrastructure layer (e.g., virtual machine or container allocation). Load balancing can reduce latency, enhance data transfer speed, and optimize resource utilization in distributed edge environments by leveraging the adaptability of SDN and the proximity of edge servers to users. In general, load balancing algorithms are classified as centralized, decentralized, or hierarchical based on the location where the load balancing decisions are made, as discussed below:

- I. **Static Load Balancing:** In static load balancing, each task assigned to a resource has a predetermined size. The system typically experiences minimal fluctuations in load, and the overall workload is distributed evenly and effectively across all available resources. However, one of the main limitations of static load-balancing methods is that tasks are executed in the exact order and format in which they enter the system. Additionally, the size and characteristics of tasks remain fixed, with no possibility for adjustments. Once assigned, tasks cannot be moved between processing units for load balancing.
- II. **Dynamic Load Balancing:** Dynamic load balancing effectively manages the distribution of workloads to ensure that a dynamic system remains efficiently balanced, especially when resources are added to or removed from it. Unlike static load balancing, Dynamic load balancing assigns or reassigns tasks to resources during runtime without prior knowledge. It also supports task migration between resources based on current load demands. As a result, the system dynamically reallocates tasks to achieve efficient load distribution.
- III. **Adaptive Load Balancing:** Adaptive algorithms represent a specialized

form of dynamic algorithms in which the algorithm's parameters and the scheduling strategy are modified according to the system's overall state. Scheduling decisions are made in real time, based on the system's current condition, allowing the workload to be distributed efficiently among available resources [52].

IV. Periodic Load Balancing: Periodic load balancing typically utilizes designated agents to collect and distribute load-related data, which helps minimize communication overhead and enhances system scalability. These policies tend to cause fewer disruptions in centralized systems than distributed ones, making them well-suited for managing load distribution in large-scale environments. Additionally, they often function effectively even when using slightly outdated information gathered between balancing intervals. Workload is redistributed among queues in the system at regular intervals to ensure optimal resource utilization.

V. Non-Periodic Load Balancing: In non-periodic load balancing, load-related data is sent to the load balancer at irregular intervals, based solely on the system's current needs. This approach is efficient in dynamic environments, as resources can be assigned to new tasks without waiting for a scheduled allocation cycle. However, this method may result in higher communication overhead, which can negatively impact the performance of the computational grid. Resource sharing is achieved by redistributing the workload among available resources in the ready queue at non-fixed intervals.

VI. Advanced Load Balancing: Advance load allocation assigns tasks to resources before they execute. This approach is critical in dynamic environments, where timely task execution can be challenging. By reserving resources for a specific period, advance allocation helps ensure smooth execution and proper load distribution. The load on each resource is assessed beforehand, and then appropriate resources are selected for task assignment to maintain balanced workload distribution [53].

VII. Centralized Approach: In a centralized approach to load balancing, a single resource within the distributed system functions as the central controller. It maintains a global perspective of the system's load status and determines how tasks should be assigned to other resources. However, as the system grows, the overhead from communication becomes substantial,

making it challenging to manage the overall system state efficiently. This can lead to the central controller becoming a bottleneck and introducing a single point of failure within the system.

VIII. Decentralized Approach: In a decentralized approach, every resource within the distributed system participates in the decision-making process for load balancing. Because these decisions are distributed, acquiring real-time global state information becomes expensive for each resource. As a result, most algorithms rely on partial data available at the local level, often leading to suboptimal but practical decisions.

IX. Hierarchical Approach: In a hierarchical model, schedulers are structured across multiple hierarchy levels. Higher-level schedulers manage large-scale resource entities, while lower-level schedulers handle smaller sub-entities. This model blends elements of both centralized and decentralized approaches to scheduling.

1.8 Fair Scheduling

Fair scheduling is used for managing computing resources that ensures CPU time, memory, and network bandwidth are shared fairly among many jobs, users, or applications. The main goal of fair scheduling is to ensure no server is overloaded or underloaded during the execution process and that jobs use all resources equally.

Fair scheduling prevents any user or process from controlling an excessive share of the available resources [54]. Fair scheduling prevents high-priority or resource-intensive jobs from managing resources, ensuring that all tasks are allocated by equal distribution or based on predetermined criteria such as task significance or limitations. This method helps maintain balance and stability within the system.

Fair scheduling is advantageous in environments with multiple users or tenants where several programs or users share the same computing resources. It ensures that no task faces resource deprivation and that all users receive consistent service levels, even with variations in workload. Fair scheduling improves system performance and overall user satisfaction by preventing resource dominance and ensuring equitable access.

1.8.1 Need for Fair Scheduling

In modern computing, especially in edge computing environments, numerous users and applications frequently require simultaneous access to shared resources, such as CPU time, memory, and network bandwidth. Nonetheless, edge devices usually possess constrained resources and must manage a variety of dynamic workloads.

Without a fair scheduling mechanism, resource allocation can become imbalanced, favoring certain tasks or users, especially those with higher resource demands or priorities. This can result in problems such as job starvation, increased latency, reduced system throughput, and in severe cases, system instability or failure. Lower-priority tasks may face indefinite delays, leading to poor user experiences and possible breaches of Service-level Agreements (SLAs).

Fair scheduling tackles these issues by guaranteeing that all tasks and users obtain an equal portion of resources, irrespective of their unique demands or priorities. By implementing rules that promote equitable resource distribution, fair scheduling enhances system performance, reduces potential resource conflicts, and ensures critical applications access the computing power they require. This method not only optimizes technical efficiency but also cultivates user trust and satisfaction through the provision of reliable and consistent service quality [55].

In situations where rapid and dependable data processing is crucial, including healthcare monitoring, self-driving vehicles, or emergency response systems, fair scheduling is vital. It guarantees that all essential tasks are carried out on time, preserving the system's integrity and responsiveness. Thus, implementing fair scheduling strategies within edge computing systems is necessary for reaching operational excellence and ensuring fair resource use.

1.8.2 Types of Fair Scheduling

In computing systems, fair scheduling denotes the equitable allocation of resources to tasks/users in a balanced and justified manner. The objective is to ensure that all processes receive rightful access to resources, preventing starvation or undue delays. A variety of approaches have been formulated to enforce equal distribution, each designed for specific scenarios based on system needs and workload attributes, and are given below:

- I. **Round-Robin:** This technique ensures even-handed resource allocation, preventing any task or application from monopolising them. This approach can result in improved resource utilization [56].
- II. **Weighted Round Robin (WRR):** This algorithm is a straightforward yet efficient method for implementing effective scheduling. It operates by allotting a weight to each edge application, which is determined by its priority level. As the priority level increases, so does the corresponding weight assigned to it. The time slice allocated to each edge application is determined by its weight, and the algorithm distributes resources in a round-robin manner [57].
- III. **Weighted Fair Queuing (WFQ):** The WFQ algorithm allocates packets to virtual queues according to their source application and priority level. The virtual queues are sequentially arranged and scheduled in a round-robin manner, considering the weight of each queue and the amount of bandwidth currently available [58].
- IV. **Proportional Fairness:** This scheduling approach assigns priority to tasks based on the current condition of the communication channel and the system's QoS requirements. Its primary objective is to maximize the system's overall throughput while maintaining fairness among tasks. To achieve this balance effectively, the algorithm often relies on accurate assessment and forecasting of channel conditions, which can involve continuous monitoring and analysis of network performance indicators such as signal strength, latency, and bandwidth availability. This ensures that tasks with higher data rate potential or stricter QoS needs are handled more efficiently without neglecting the rest [59].
- V. **Deadline-Aware Scheduling:** Some systems incorporate task deadlines into the fairness model. The scheduler considers both the fairness of resource distribution and the deadline of each task. Priorities are given to the time-sensitive tasks without completely ignoring less urgent tasks. Each of these scheduling types serves a specific purpose. The choice of algorithm depends on the system architecture, the nature of the tasks, and the performance goals of the computing environment.

Fair scheduling means allocating computational activities and system resources fairly across users, applications, and edge devices in edge computing. This approach addresses system norms, responsiveness, and user satis-

faction. However, incorporating fairness into scheduling algorithms in distributed, limited resource environments poses significant design problems due to the many constraints.

1.8.3 Benefits of Fair Scheduling in SDN-based Edge Computing

Some benefits of fair scheduling that help improve the QoS are discussed below:

- **Balanced Resource Allocation:** Fair scheduling implements uniform distribution at the edge nodes so that no single device is overloaded to the detriment of others being underloaded. This results in improved overall resource usage, lower energy cost, and better system throughput. This is mainly useful in systems facing dynamic workloads and having limited processing capabilities [60].
- **Enhanced Service Continuity in Compliance with QoS:** Fair scheduling maintains a predetermined, well-defined service level through the balanced allocation of processing, storage, and bandwidth. It also reduces end-to-end achievable throughput and high-severity blockages and guarantees adequate support for time-sensitive functions where high priority is needed.
- **Improved User Experience:** Equitable resource access policies ensure that all users, irrespective of their geographical region and access devices, receive the same quality of service. Service uniformity enhances user perception of system performance and confidence in the system. This is crucial in sensitive areas such as healthcare, autonomous systems, and intelligent infrastructure monitoring.
- **Fair Resource Management from a Multi-Client Perspective:** When edge computing resources are shared across multiple users/organizations, fair scheduling strategies make it impossible for any user to monopolize resource use. This meets the SLA requirements and enables fairness and scalability for service provision to diverse clients and applications [61].
- **Low Energy Consumption:** Balanced resource distribution prevents specific devices from becoming overloaded, improving energy efficiency by reducing excessive power usage. Proper task allocation is vital for sustainabil-

ity in edge environments with limited energy, processing capabilities, and computational resources.

1.8.4 Challenges of Fairness Scheduling in SDN-based Edge Computing

Some challenges are being faced by fairness scheduling when implemented in SDN-based Edge Computing.

- **Differentiation in Resource Providers:** The edge system comprises components with different computing power, memory, and varying degrees of connectivity. Maintaining fairness in heterogeneous hardware is often complex. Uniform workload distribution can increase device utilization, which demands device-sensitive adaptive scheduling systems.
- **Immediate Demand for Latency:** Real-time applications like vehicle monitoring and control or augmented reality need instant response times. Strict adherence to fairness perspectives often postpones critical tasks, especially when resources are equally partitioned. In other words, there is a conflict between fairness and the need for immediacy. Therefore, latency-sensitive scheduling strategies that balance the two extremes are needed.
- **Sudden and Intense Load Theory:** Localized edge environments commonly have intense and highly unpredictable workloads. Achieving fairness in allocation under such circumstances is extremely difficult since allocation schemes that were previously considered best can quickly become wastefully suboptimal. Having to reallocate or assign tasks frequently strains system resources and may lead to periods of reduced efficiency or increased latency.
- **Limitations of Resources:** Edge nodes are inherently constrained compared to centralized cloud infrastructure. This is evident with edge nodes having less computing power, storage, and memory. The attempt to enforce fairness among these scant resources inevitably leads to inefficiencies or task refusal, which can be deemed irresponsible. Achieving resource allocation equity amidst system productivity becomes increasingly straining.
- **Mobility of the User and Device:** Users and devices in mobile edge systems and networks are in constant motion, meaning mobility is a core feature in many edge use cases. These movements may change task foci and

necessitate constant scheduler workload evaluation and reallocation. Due to the nature of these systems, attempting to achieve and maintain fairness without severely restricting efficiency or response time is tremendously difficult.

- **Security Risks and Shared Set Environments Complexity:** Multi-tenant setups make user behaviour unique in an edge network. Users can have differing, even opposing, priorities and usage patterns or be outright hostile. A lack of proper governance in fair scheduling can cause resource misallocation. Adding security, monitoring, and control-access mechanisms into the scheduling process causes performance degradation because smooth performance is no longer achievable.

1.9 Fair Scheduling and Resource Optimization

Resource optimization is critical in any computing system, especially in environments like edge computing, where resources are inherently limited. It involves maximizing resource usage efficiency while ensuring system performance remains within acceptable limits. Fair scheduling contributes directly to this goal by promoting balanced and intelligent use of available resources.

When tasks are scheduled fairly, no system component is overloaded while others remain underutilized. This leads to a more uniform distribution of processing loads, which helps prevent bottlenecks and avoids the need for frequent task migration or rescheduling. Fair scheduling also reduces the chances of task rejections, ensuring that all tasks, including low-priority ones, can be executed [62].

From an optimization perspective, fair scheduling helps achieve better throughput, minimize response time, and enhance the overall quality of service. It allows edge computing platforms to support more users and devices without compromising service reliability or speed. Moreover, it supports energy efficiency by reducing idle times and minimizing unnecessary task switching, which is especially important in power-sensitive edge environments. In short, fair scheduling is not just about fairness. It is a foundational strategy for optimizing resource management and maintaining system health in dynamic computing infrastructures [63].

1.10 Fair Scheduling in SDN-based Edge Computing

In SDN-based edge computing, managing tasks from various devices across multiple locations necessitates a thoughtful and equitable scheduling approach. Given that edge environments have limited computing power and bandwidth, a fair scheduling system must ensure that all tasks receive appropriate resources regardless of size, urgency, or source [64]. Specific applications or users may dominate edge resources without a fair mechanism, causing delays, performance degradation, or complete task rejection for others. This leads to poor load distribution, decreased system reliability, and reduced user satisfaction. Integrating SDN into edge computing provides centralized control and global network visibility to tackle these challenges. SDN separates the control plane from the data plane, allowing a centralized controller to make real-time, intelligent decisions regarding resource allocation, traffic routing, and task scheduling based on the overall state of the network. However, implementing fair scheduling in SDN-based edge environments introduces several challenges. The diversity of edge servers and the dynamic nature of workloads influenced by fluctuating data rates, mobile users, and evolving service requirements complicate the design of universal scheduling policies. In multi-tenant scenarios, where different users or services share the same edge infrastructure, maintaining fairness while ensuring performance isolation becomes even more intricate. Despite these challenges, SDN provides powerful tools to address them. With centralized control, SDN can continuously monitor resource usage across the edge, enforce load balancing policies, dynamically adjust scheduling priorities, and adapt to real-time network conditions.

The advantages of fair scheduling in this SDN-assisted environment are significant. It prevents resource monopolization, ensures balanced workload distribution across all nodes, and helps avoid performance bottlenecks. This leads to improved system responsiveness, reduced processing delays, better compliance with service-level agreements, and enhanced quality of service for all users [65]. Most importantly, it ensures optimal resource utilization by fully utilizing available computational resources without overloading any single node.

A close look at the current research shows that although many studies have focused on essential performance measures like response time, latency, and throughput, the main issues of fair scheduling and resource optimization, especially makespan, average resource use, and load balancing, have not been studied

as much. These parameters are crucial for evaluating the efficiency and fairness of scheduling strategies in highly dynamic and resource-constrained edge computing scenarios. Since SDN is key to creating flexible, real-time, and rule-based scheduling, more research is needed to develop fair scheduling methods that use SDN features to better use resources, provide equal access, and improve the overall performance of edge computing systems.

1.11 Motivation

Integrating SDN with edge computing has emerged as a promising paradigm for managing distributed computational resources efficiently. However, existing scheduling algorithms, such as First-Come-First-Serve (FCFS) [66], Shortest Job First (SJF) [67], and Earliest Deadline First (EDF) [68], exhibit significant limitations in dynamic and heterogeneous edge environments. For instance, while Dynamic Weighted Edge Computing Scheduler (DWECS) [69] integrates FCFS, SJF, and EDF for task scheduling, it lacks an effective server selection mechanism, adversely impacting overall system performance. Similarly, SJF's preference for shorter tasks leads to the starvation of larger tasks, degrading QoS metrics like makespan and overall completion time. FCFS does not account for server load conditions, resulting in inefficient resource utilization.

Advanced algorithms like Parallelization Task Allocation and Load-balancing algorithm (PTAL) [70], and Assigning Computing Tasks among multiple Edge Computing Servers (ACTE) [71] algorithm attempt to address load balancing and task allocation, but fall short by not considering task sizes or implementing comprehensive load balancing strategies. Min-Min and Max-Min [72] algorithms aim to use resources efficiently; however, Min-Min tends to favour smaller tasks, leading to underutilization of available resources. Max-Min is more balanced but doesn't completely handle the changing conditions of edge environments. Additionally, methods to improve performance, such as the Optimal IoT Service Offloading (OSO) [73] approach that uses the Non-dominated Sorting Genetic Algorithm-II (NSGA-II), have been suggested to manage uncertainties in SDN-based mobile edge computing. While OSO focuses on optimizing completion time and load balance variance, it faces challenges addressing dynamic uncertainties such as network congestion and edge node performance degradation. Additionally, the computational overhead introduced by NSGA-II raises concerns about scalability and adaptability in large-scale edge environments.

Similarly, commonly used for resource allocation, the First Fit (FF) [74] algorithm assigns tasks to the first available resource without considering edge nodes' current load or heterogeneity. This approach can lead to suboptimal resource utilization, as it may cause specific nodes to become overloaded while others remain underutilized. The lack of consideration for server heterogeneity, load balancing, and failure management limits its applicability in complex systems.

The Utility-Aware Task Offloading (UTO) [75] algorithm, which involves multiple access points collecting data and sending it to servers, lacks a robust mechanism for handling the massive influx of data from various sources. This deficiency leads to underutilized and overutilized machines, adversely affecting the overall resource utilization of the system.

Particle Swarm Optimization (PSO) [76], inspired by the social behavior of bird flocks, has been applied to task offloading and resource allocation in edge computing environments. While PSO offers advantages in terms of convergence speed and simplicity, it often suffers from premature convergence and cannot effectively handle edge networks' dynamic and heterogeneous nature. Moreover, PSO's performance can degrade in high-dimensional search spaces, making it less suitable for complex scheduling scenarios in SDN-based edge computing.

Researchers have also utilized GWO [77], which is based on the leadership hierarchy and hunting mechanism of grey wolves, for task scheduling in edge computing. Although GWO demonstrates strong exploration capabilities, it may struggle with exploitation, resulting in suboptimal convergence in some cases. Additionally, GWO may not sufficiently address the multi-objective nature of resource allocation problems, such as balancing energy consumption, latency, and load distribution, which are critical in edge environments.

Given these challenges, there is an urgent need to develop a novel framework that integrates fair scheduling with resource optimization tailored for SDN-based edge computing. This framework should address the limitations of current algorithms by considering the size of tasks, the load on servers, and fluctuating network conditions to enhance quality of service factors such as average resource usage, total task completion time, and workload distribution. The proposed solution employs a central control mechanism and a comprehensive view of the SDN network to ensure equitable resource sharing, prevent task stalls, and adapt to the dynamic conditions of edge computing environments, ultimately enhancing the system's performance and reliability.

The following needs are crucial for creating a fair and effective scheduling and resource optimization system to address the challenges of current resource optimization algorithms in SDN-based edge computing.

- The system must dynamically accept and process task requests at any time, regardless of task size, resource status, or current workload. Using SDN's central control and overall network view, the system should adjust to current network situations, ensuring tasks are shared effectively and resources are used as much as possible across different edge nodes.
- The framework should minimize task execution time and processing costs while optimizing various QoS parameters, including latency, throughput, makespan, and average resource utilization. Using SDN's ability to be programmed, the system can set rules that prioritize tasks according to their QoS needs, ensuring that essential applications get the resources they need quickly.
- The system should perform efficiently while increasing the number of devices and applications. The integration of SDN and edge computing provides a flexible architecture that facilitates the new edge nodes and IoT devices while affecting the others.

1.12 Research Objectives

The subsequent objectives have been established to investigate the research problem systematically.

- I. To study and analyze the existing resource optimization techniques in the SDN-Based Edge Computing Environment.
- II. To develop a fair scheduling resource optimization strategy for enhancing average resource utilization.
- III. To optimize scheduling parameters for the proposed approach under the SDN-Based Edge Computing Environment.
- IV. To evaluate and validate the performance of the proposed techniques with standard metrics.

1.13 Thesis Organization

The thesis is structured into six chapters as shown in Figure 1.4. Chapter 1 outlines the research background, the significance of edge computing and the integration of SDN with it. This chapter also highlights the various aspects of resource optimization and the motivation behind the study. Chapter 2 presents a detailed review of existing literature on resource allocation, load balancing, and task scheduling in SDN-based edge computing, emphasising the research gaps. Chapter 3 introduces a hybrid PSO-GWO algorithm for effective resource allocation. Chapter 4 proposes an average-based task scheduling, resource allocation and load balancing strategy designed for SDN-enabled edge environments. Chapter 5 describes a dynamic, priority-based task scheduling method to enhance resource optimization in SDN based edge computing. Finally, Chapter 6 summarises the main findings and offers suggestions for future research.

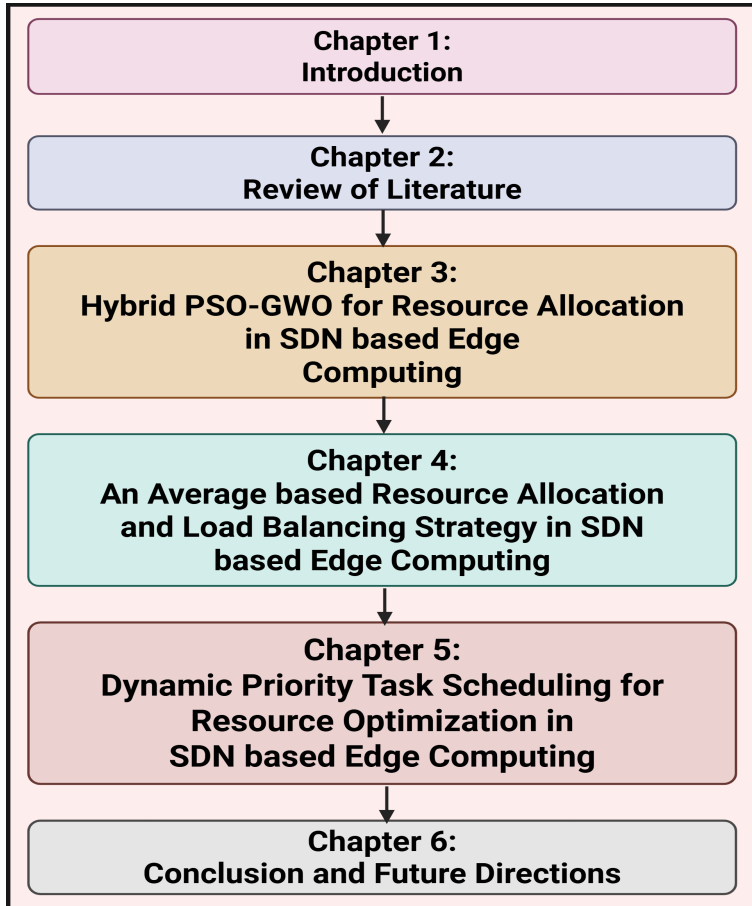


Figure 1.4: Flowchart of Thesis Organization

Chapter 2

Review of Literature

A detailed review of existing research is crucial for supporting the advancement of SDN-based edge computing and bridging the gap between current solutions and emerging requirements. This chapter presents an in-depth analysis of existing methodologies applied in Edge and SDN-enabled edge environments, focusing on resource allocation, scheduling, and load balancing techniques. The assessment focuses on identifying the main limitations and performance issues of these approaches. Despite considerable progress in the domain, critical challenges persist, especially in ensuring efficient coordination between edge nodes, maintaining low latency, and achieving high availability in dynamic network conditions.

A visible gap is observed in the research about combining SDN in better ways to optimize resources in edge computing, such as resource allocation, intelligent scheduling, and load-balancing methods designed for SDN-controlled edge networks. Addressing these challenges is crucial for enhancing the responsiveness, reliability, and effectiveness of distributed edge systems. In this chapter, we examine these limitations and propose a framework that can improve the operational efficiency of SDN-based edge computing.

The objective is to examine the impact of SDN integration with edge computing. Attention is given to integrating SDN and edge computing environments, particularly in terms of how they interact with key resource provisioning mechanisms to ensure system reliability and enhance QoS. The literature is further

analyzed to explore how these mechanisms perform under dynamic, distributed, and heterogeneous edge environments, typical of real-world scenarios. Additionally, this chapter presents an extensive literature review with a multidimensional objective tailored to SDN-based edge computing environments. The primary goal of this study is to examine existing techniques for resource optimization in edge computing, with particular emphasis on computation offloading, dynamic resource allocation, load balancing, and task scheduling.

SDN-based edge computing has transformed the management, allocation, and utilization of computing resources, particularly in real-time applications that demand rapid and effective processing. By placing computing power closer to where the data is generated and using SDN's centralized control, edge computing systems can respond faster and better manage resources. This architecture is particularly beneficial for applications that require low latency and real-time performance. Researchers have suggested several strategies to optimize available resources, focusing on essential elements like resource allocation, task scheduling, load balancing, and computation offloading.

These techniques aim to allocate the tasks to the most suitable edge nodes, balance network traffic, and optimize processing efficiency. SDN supports achieving these goals by providing a comprehensive perspective of the network, offering flexible resource allocation, and enhancing decision-making. Therefore, edge computing that utilizes SDN can improve system performance, minimize delays, and guarantee a more effective use of network and computing resources.

2.1 Research Methodology and Plan of Study

This section outlines the methodological framework used to conduct a qualitative analysis of the literature within the review, as well as the sources of the selected state-of-the-art contributions. It also includes the methodology integrated into the proposed research and discusses the contribution to the conducted review. According to Prisma 2020, Kitchenham and Charter's 2007 articles, the Systematic

Literature Review (SLR) methodology is employed in the current investigation [78-79]. Our objective is to comprehensively evaluate various research papers and examine the contributions of prior researchers, providing a methodical review that can assist fellow researchers. This section focuses on and indicates the sources of the existing state of the art, such as data sources and different research requirements.

2.1.1 Nature of Searching

A systematic survey of resource optimization in edge computing and SDN-based edge environments was conducted through well-known sources. Several search keywords include optimization in edge environment, resource optimization in SDN-based edge environment, overview of SDN-based edge paradigm, resource provisioning in SDN-based edge environment, dynamic resource allocation, and resource allocation in SDN-based IoT-Edge management.

2.1.2 Sources of Data

Numerous reliable sources have been selected for reviewing and surveying this entire literature to compile the various methods and approaches employed in this study. Additionally, all journal and conference papers are found using databases such as Scopus, WoS, SCI, SCIE, and others. The papers are carefully examined to find all the applicable fields where the suggested techniques are implemented. Some of the frequent sources are given in Table 2.1:

Table 2.1: Frequent Sources of Manuscript

Elsevier	https://www.elsevier.com
Springer	https://springer.com
IEEE	https://ieeexplore.ieee.org
ACM Digital Library	https://dl.acm.org
Wiley Online	https://onlinelibrary.wiley.com

The statistics of the selected papers according to the percentage of published articles by various publishers is presented in Figure 2.1. It is evident from the

figure that the major portion of published sources belong to IEEE . Springer is the second major source for related articles. Remaining sources are from Wiley, ACM and Elsevier.

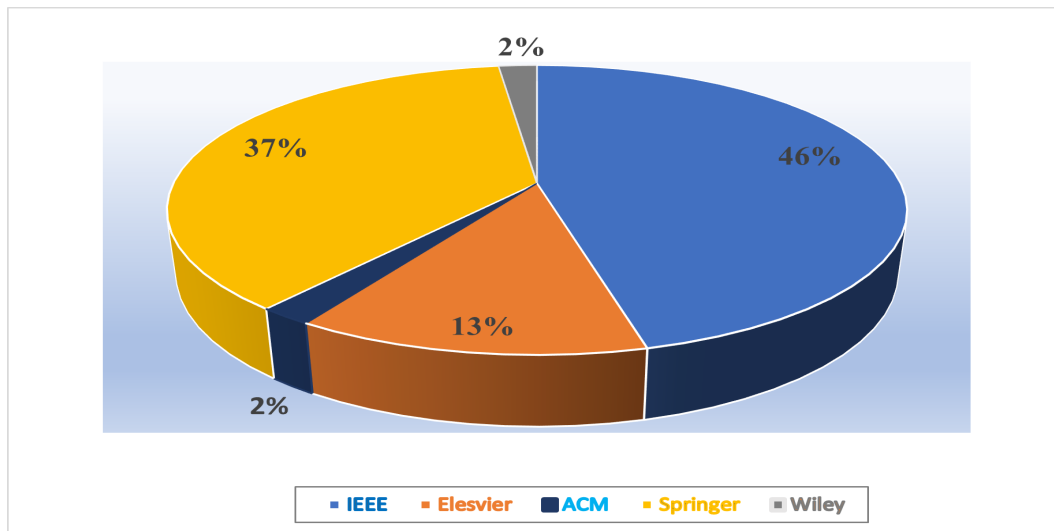


Figure 2.1: Percentage of Manuscripts from Frequent Sources

2.1.3 Duration and Validity of Study

This literature survey primarily comprises papers from 2013 to 2025, selected from reputable journals, books, and conferences. The statistics of the included years of publications from 2013 to 2025 are presented in Figure 2.2.

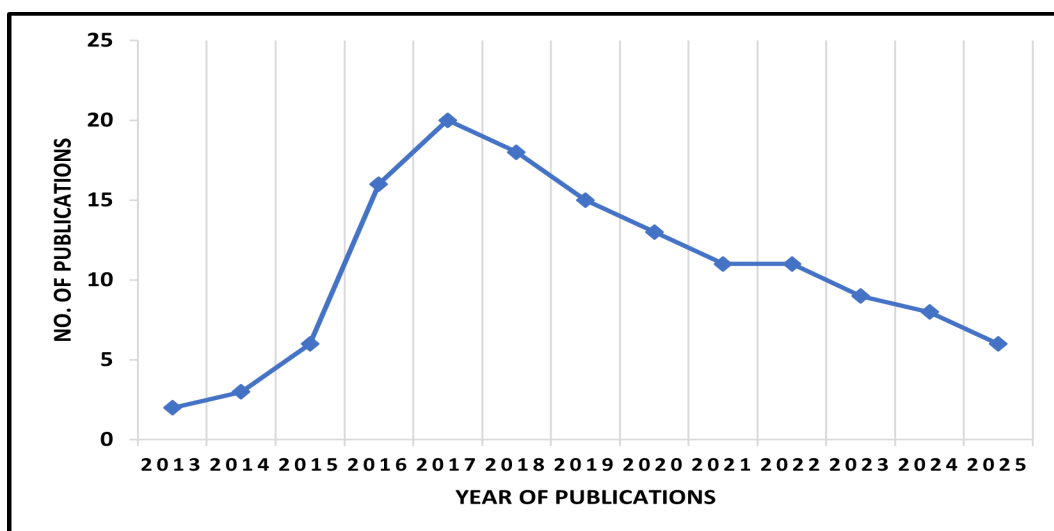


Figure 2.2: Number of Articles (Year Wise)

The plotted graph reveals that the highest number of manuscripts was published between 2016 and 2019. The period from 2012 to 2019 marked a significant boom in publications within the relevant fields. However, this figure gradually decreased after 2019.

2.1.4 Criteria for Inclusion and Exclusion

Our sole basis for considering articles (Inclusion) is whether they meet the selection requirements. The exclusion criteria serve as the foundation for omitting research papers from the study. Figure 2.3 presents a flowchart outlining the Inclusion and Exclusion Criteria for paper selection.

Inclusion criteria:

- Articles composed in the English language.
- The search strings are pertinent to the subject matter at hand.
- A research paper that can be verified through reliable sources.
- The research articles that are published in reputed Journal/Conferences/Books.
- The publication dates of the materials under consideration range from 2012 to 2025.
- The papers must satisfy the criteria of nature of searching as discussed in section 2.1.1.

Exclusion criteria:

- The discussion lacks focus on the pertinent concerns surrounding the IoT, Edge, and SDN technology.
- An enough attention was not given to the sole optimization issues posed by combining SDN-Edge Technology with the IoT.
- The search criteria were not met as mentioned in section 2.1.1.
- The focus of the article on the unrelated domains.
- The article offers no new insights, methodologies, or approaches related to the convergence of SDN and Edge Computing.
- The article has not undergone peer review.
- The publication predates the year 2010.

Flow chart

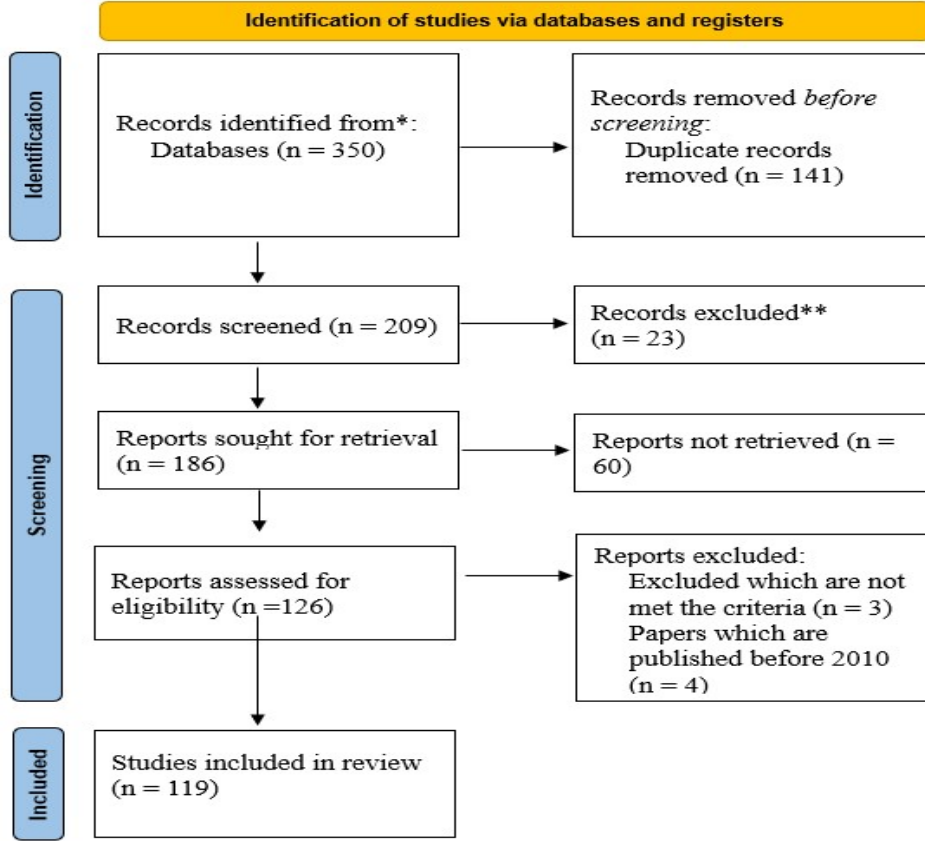


Figure 2.3: Flow Chart of Inclusion Exclusion Criteria

2.1.5 Review Judgement

The review judgement focuses on the background of the methods used to perform qualitative judgment in the review. The inclusion and exclusion of the articles published were based on certain aspects. The selection of the related articles was done only after analyzing the abstract. After article selection, the article was thoroughly reviewed, and inclusion and exclusion criteria were applied based on standards as mentioned in Figure 3. Besides, the inclusion of articles based on the following conditions:

- Resource optimization in edge environment and integration of cloud-edge with SDN.
- Different architectures in edge and SDN paradigms. Also, the proposed articles were critically analyzed based on the techniques, findings, optimized parameters,

shortcomings, etc.

2.1.6 Adopted Methodology

Various methods were utilized in our comprehensive review process to gain an all-encompassing understanding of research in the related field. Five electronic databases were searched using specific search terms, which were then screened for eligibility based on predefined criteria by at least two reviewers. Full texts that met our standards were then reviewed in depth while extracting pertinent data from eligible studies via standardized forms—findings that underwent synthesis through narrative approaches and meta-analytic techniques when applicable—and received quality assessments utilizing standardized tools upon inclusion into our analysis pool. Additionally, a hand-search of reference lists and citation searches were conducted to ensure further comprehensiveness. The inherent value of this methodology is the ability to identify and analyze the most relevant and rigorous studies, ultimately providing an extensive overview of research in the field. As a final step, the findings are summarized clearly and concisely, providing an overview of the state of the research. Graphical interpretation of the steps is shown in Figure 2.4.

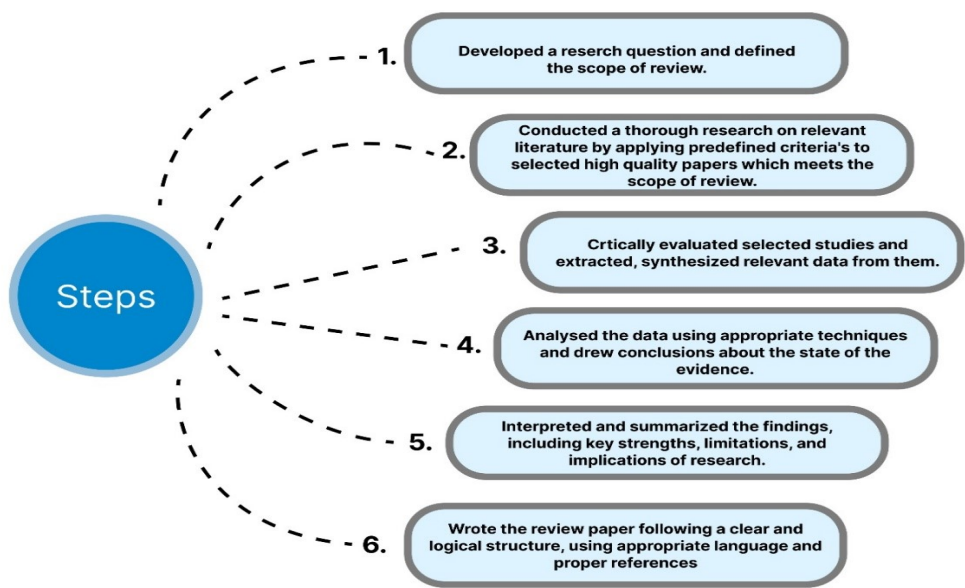


Figure 2.4: Schematic Diagram of Adopted Methodology

2.2 Related Work for Resource Optimizations Aspects in Edge Computing

This chapter aims to review recent studies on load balancing, resource allocation, task scheduling, and computation offloading and conduct a more comprehensive analysis of several related literature studies [80]. The technical terms used in edge computing and Mobile Edge Computing (MEC), serve the same purpose and have identical meanings. Furthermore, these technologies consistently maintain their integrity and generality. Several comprehensive studies have focused on various aspects of resource optimisation in edge and mobile edge computing.

MEC is an ingenious technology that can significantly enhance the performance of computing-intensive services, such as Augmented Reality (AR) applications and IoT, by bringing data processing and computation tantalisingly close to end-users. This reduces latency and amplifies the QoE for users, breathing new life into the entire gamut of services. Numerous resource allocation plans have been published in the literature to optimize the use of communication and computational resources [81].

The author Comprehensively analyzes current research on computation offloading at the edge [82]. It discusses various aspects of Computation offloading, including ways to reduce energy consumption, improve the quality of experience, and ensure quality of service. Furthermore, the review offers valuable insights into various approaches to resource allocation.

The author provided a thorough overview of task offloading strategies for MEC that other researchers have examined [83]. A literature review was conducted on current studies on task offloading for MEC. Three groups of models have been identified: algorithm paradigm, decision-making, and computational models. Another goal of this work was to classify and clarify the architectural model employed in task offloading. The practical use cases, research directions, and design issues associated with this field are also discussed.

The authors [84] discuss different methods for scheduling resources collaboratively within the edge computing framework. They also discuss task offloading, resource allocation, and resource provisioning as the three main scheduling research challenges in edge computing, and compare techniques for scheduling centralized and distributed resources.

2.2.1 Resource Allocation and Scheduling in Edge Computing

Resource Allocation and Scheduling are integral concepts of various decentralized domains, including computer networking. They involve the strategic distribution of resources to achieve better resource utilization and the maximum performance and efficiency of the system. They are crucial in managing tasks, processing data, and workflow.

Resource allocation in edge computing includes approaches incorporating application placement, task offloading, load balancing, and resource provisioning [85]. Several research studies focus on optimizing resource allocation using different models and frameworks. In the edge environment, many Edge Servers (ES) are available to handle service requests from IoT devices. Every service request can be divided into a set of tasks. To achieve a SLA, the primary objective of resource scheduling in edge computing is to identify the optimal approach for allocating various tasks to edge nodes, ensuring efficient completion.

The author [86] devised a revolutionary MEC-based resource allocation strategy for IoT applications. This cognitive edge framework seamlessly blends dynamic resource allocation with edge computing resource demand mapping to efficaciously support the enterprise cloud, resulting in a phenomenal outcome. The cloud operating system's bidirectional resource sharing creates a single framework for allocating resources for IoT, leading to a harmonious coexistence of technology and innovation. The test results demonstrate beyond doubt that the proposed technique is far more efficient in handling resource allocation requests, further re-

inforcing its position as a game-changer in computing.

Ali and Simeone [87] have proposed a unique MEC-based resource allocation strategy for AR applications. The plan successfully reduces system costs by combining communication and optimizing IT resources. A consecutive convex approximation function ensures optimal energy use in the MEC. The results demonstrate that the method produces superior load reduction capacity compared to traditional methods.

Another integration of edge computing and IoT is proposed in [88]. It describes cross-layer dynamic network optimization to improve the network setup's performance. The authors used the Lyapunov dynamic stochastic optimization approach for admission control and better resource allocation. The author of [89] describes the challenges edge computing faces regarding resource management and utilization. The author also designed a model for improved resource allocation in edge environments by introducing a unique key within the system.

According to [90], Wi-Fi access networks have been used to improve MEC services. The results of a resource management strategy based on two-layer time division multiplexing demonstrate that MEC over Wi-Fi can utilise power reliably. It is suggested to use MEC-enabled Fibre-Wireless broadband for low-latency, demanding MEC applications [91], which is the basis of this work. A cloud-enabled resource management scheme has been proposed to reduce network latency. The results demonstrated that the analytical model successfully reduced delay and saved energy.

The DRAP solution [92] utilises a middleware layer to connect mobile devices and peripherals. Any mobile device that needs resources can access a pool of resource-rich edge devices. Resource discovery, idle resource computation, and mobile and peripheral functions are all handled by DRAP. This method is very flexible because no device acts as an edge device, and because it can automatically reorder as nodes are added or removed. Besides, to minimize latency and energy

consumption and maintain load balancing within Edge Infrastructure Providers' (EIPs') budgetary constraints, the author in [48] investigates optimizing computation offloading, task migration, and resource allocation in MEC federation using an MDs clustering algorithm and a decentralized deep deterministic policy gradient-based framework.

[47] The author proposed a P2P system to overcome the problem of selfish behaviour between participating devices. Under the point-based incentive mechanism of this system, devices can share resources to earn points, which they can then use to purchase resources from other devices. The system is built on a pre-trust model, considering the social obligations of the community group to ensure that participating devices are trusted. A middleware framework is proposed in [93] to improve CPU usage for AR applications. The platform offers software services for augmented reality applications, including functionality to deploy or remove software components at runtime, utilising only those necessary for the specific application. While this reduces additional demands on the CPU, the technique only applies to AR applications.

Table 2.2: Related Work on Resource Allocation and Scheduling

Author, Year & Refer- ence	Findings/ Outcomes	Limitations	Research Contri- bution	Results	Optimized Param- eters
Amjad, 2017 [86]	The primary advantage of edge computing is the decrease in latency.	Edge devices have less processing, storage, and computational capability than centralized data centers	Edeg computing based IoT application resource allocation framework.	The proposed system manages resource allocation requests more efficiently.	Cost, Throughput

Ali & Semione, 2017 [87]	Edge computing offers improved reliability by reducing dependency on a centralized infrastructure.	Managing and growing edge computing infrastructure gets increasingly difficult as edge devices and applications proliferate.	Resource allocation for AR applications using energy-efficient MEC.	The proposed method achieves better load reduction than the current methods.	Energy
Shichao, 2018 [88]	The purpose of utilizing computational resource management is to enhance the overall performance of communication systems.	Interference Management, Didn't Show Proper Effective energy harvesting.	Mechanism for the cross-layer distribution of dynamic resources.	The proposed scheme achieves improvement in performance and throughput.	Performance, Delay, Throughput
Jinlai, 2017 [89]	Resource-sharing contracts with edge server providers and employ resource provisioning and latency-aware scheduling.	Showing Total dependency on the server in micro data centers.	A zenith resource allocation model.	A good resource allocation scheme helps to improve performance.	Response Time, Performance

Rimal, 2017 [90]	Edge devices and servers may use encryption and access restrictions to safeguard data from breaches and illegal access.	Specialized frameworks and protocols are needed to handle edge devices, servers, and their interfaces with centralized cloud infrastructure.	Use of Fiber-wireless (FiWi) access networks for optimizing MEC services.	With MEC over WiFi, the proposed strategy has demonstrated adequate power consumption.	Delay, Response Time, Energy
Rimal, 2017 [91]	Edge computing may also work in low-connectivity contexts. Applications may share edge computing resources.	Reduced bandwidth costs and efficiency may offset these early expenditures over time.	Arrangements for managing resources within the MEC via a wireless fiber optic network.	Both energy efficiency and quality of experience were improved by the proposed method.	Response Time, Energy
Zhenquan, 2019 [94]	Minimize the task execution latency and power consumption's weighted sum cost as achievable.	The results are based on ϵ -greedy Q-learning.	The proposed algorithm for task offloading is an optimal approach that utilizes the ϵ -greedy Q-learning technique.	Low-Power Consumption, Low task execution latency.	Resource allocation, cost, performance, energy.

Liu, 2020 [95]	Efficient resource allocation with the lowest cost	Compared with the probabilistic offloading scheme, Heuristic algorithms are not considered.	An iterative optimization algorithm for proficient resource allocation.	Increased resource utilization and minimized the cost and time consumption.	Cost, Performance
----------------	--	---	---	---	-------------------

2.2.2 Load Distribution and Balancing in Edge Computing

Load balancing across edge devices is a critical aspect of edge computing, which is becoming increasingly important as the use of linked gadgets and IoT devices continues to grow. Instead of processing data in a central cloud, edge computing processes data at the network's edge, lowering latency and enhancing the network's performance. Edge load balancing is a fundamental factor in ensuring efficient data processing. Researchers have explored various methods for allocating resources, such as CPU and memory, to different tasks and applications to maximize performance and minimize downtime.

Coo Load, a collaborative load-balancing system created by Beraldi [96], is positioned to reduce execution latency at the network edge. Under this collaborative system, data centres exchange cache space based on available space. If the cache at the data centre becomes saturated, incoming requests are redirected to another data centre with sufficient storage capacity. Through experimentation, this solution has been proven to enhance the efficiency of the information technology department significantly.

Load balancing is achieved by assigning VMs to edge devices that have sufficient resources, as proposed in [97]. The primary objective is to reduce travel time while considering WAN bandwidth and the load on edge devices. To alleviate workload, the method described in [98] integrates mobile devices, edge devices, and

remote clouds. A mobile device in need of resources can connect to a nearby edge device or a remote cloud. Low latency and no internet bandwidth are required when the service is provided through an edge device. However, if the service is sourced from a remote cloud, it follows a typical MCC pattern with higher latency due to internet bandwidth usage.

The predefined VM templates are utilized per the approach suggested by [99] to fulfil user requirements. To maintain infrastructure stability, the edge device selects the optimal template, customises the infrastructure, and then rolls back the changes. Using VMs separates infrastructure-level changes from modifications to guest operating systems; however, load balancing and task sharing between tenant VMs are not addressed.

Consistent with MEC, Kumar [9] created an intelligent network data management system for a network that can withstand vehicle delays. A VM migration strategy reduces data centre power consumption. Electric cars at the network's edge handle handling and communication issues while making independent judgments. The test results demonstrate that the proposed solution provides higher throughput and lower network latency.

Table 2.3: Related Work on Load Balancing and Distributions

Author, Year & Refer- ence	Findings/ Outcomes	Limitations	Research Contribu- tion	Results	Optimized Parame- ters
Kumar, 2016 [9]	Edge com- puting lets programs run with- out a cloud connec- tion.	Skills and resources may be needed to maintain a distributed edge net- work's reliability, security, and effi- ciency.	Data man- agement strategy for MEC Smart Grid with media latency tolerance.	The pro- posed solution increases through- put while minimizing network latency.	Energy, Delay, Response time, Through- put

Beraldi, 2017 [96]	Edge computing keeps sensitive data closer to its source and reduces transmission exposure, improving privacy and security.	Distributed processing makes maintaining data consistency and integrity across devices more challenging.	Collaborative load balancing arrangement at MEC.	The performance of IT services is greatly improved by the proposed approach.	Time Delay, Average Load
Yaozhong, 2017 [100]	Load balancing Within edge computing based on QoS parameter	The priority of tasks is not taken into consideration.	Distributing new tasks that come into the edge computing network at regular intervals	Effective Load Balancing And improved overall performance and output.	Makespan and Performance
Tham, 2017 [101]	The objective is to reduce the total processing time of the application.	Energy-constrained are not taken into consideration	Presented a load balancing system that is both distributed and capable of achieving quick convergence.	The objective is to reduce the total processing time of the application.	Performance, Throughput, Fault tolerance, Reliability

Babou, 2020 [102]	Distribute the requests hierarchically on the HEC clusters and reduce the latency.	The automation of resource management.	Hierarchical Load Balancing technique	Effective load Balancing with Improved processing time of request and reduced delay	Latency and Performance
Kadhim, 2021 [103]	A Load balancing approach to improve utilization of resource	Energy constraints are not taken into consideration	A Load Balancing scheme between Edge servers	Maximizing the CPU utilization, Priority scheduling, Minimizing the required bandwidth	Bandwidth consumption, Latency, Response time, Resource utilization
Zhang, 2020 [104]	Secure load balancing and computation offloading algorithm	As mentioned in the article no security parameters are considered	Load balancing algorithm for distributing mobile data used among small base station	Reduce overall communication cost	Performance Cost

2.2.3 Computational Offloading and Rescheduling in Edge Computing

Computational offloading is an essential concept in resource optimization. It describes how computational tasks are managed to use resources effectively. It transfers workload from local devices to remote sources, helping to achieve maximum system efficiency. This approach is instrumental in enhancing resource utilization, improving efficiency, and minimizing task execution time. It works in various fields, including edge computing and MEC, as well as cloud computing. Here, we discuss various computation offloading schemes in edge environments.

Chen [49] has proposed a game theory strategy to investigate the issue of offloading in mobile cloud computing. The system creates a distributed multi-user compute offload set among mobile device users to achieve optimal computing in the network. Game engineering successfully achieves the Nash equilibrium property and reduces the computational load for many users.

Laredo [105] presented an important self-organization technique based on Bak-Tang Wiesenfeld's automated sand cell model for pressure equalization for computational tasks. Experimentation has revealed improved resource utilization without any degradation in the QoS.

Author Li [106] created a low-latency MEC design in mobile media networks. The system uses an MEC server and an optimized traffic scheduling mechanism. As a result, a modified handover technique improved mobility management. For task scheduling in MEC systems, Liu [15] suggested a Markov decision method that reduces the average latency and power consumption. Comparing the results with reference techniques reveals minimal delay.

A computationally offloading approach for MEC systems was created by [107, 108] using portable Energy Harvesting (EH) equipment. The model uses a dynamic computational offloading method based on Lyapunov optimization, and the outcomes demonstrate that it outperforms benchmark policies. Various research papers have suggested different advanced computational techniques. [109] proposes a Centralized Enterprise Cloud (CEC) model, where all mobile and edge devices register with the edge computing, which assigns the appropriate edge device based on resource requirements. The advantage is that work can be continued on multiple peripherals, saving money, time, and energy. However, this plan may not be scalable to larger networks because it relies too much on internet capacity. A similar strategy is presented in [110], which utilizes a centralized origin server that maintains records of connected peripherals and their associated services. The origin server forwards requests to the appropriate edge device, which can perform the task and distribute it to multiple edge devices.

Table 2.4: Related Work on Computational Offloading and Rescheduling

Author, Year & Refer- ence	Findings/ Outcomes	Limitations	Research Contribu- tion	Results	Optimized Parame- ters
Chen, 2016 [49]	Edge computing processes data locally and responds instantly, enabling real-time decision-making.	Installing edge devices and networking infrastructure requires money.	Mobile Cloud Offloading Method Using Game Theory.	The performance of the proposed unloading system is improved.	Average Utilization, Computation Offloading
Laredo, 2017 [105]	Only necessary data is transferred to the cloud or data center by processing and filtering data at the edge.	Edge computing processes and stores data closer to its source, lowering cloud-based visibility and management.	Important self-organizing method for load balancing in MEC.	Better resource consumption is demonstrated by the proposed method without sacrificing QoS.	Execution time, Average Load, Load Balancing
Li, 2017 [106]	Distributed edge computing offers horizontal scalability.	Configuring, monitoring, and maintaining a distributed edge device network is difficult.	MEC design in low-latency mobile vehicle network.	Results showed enhanced mobility management with the intercell handover mechanism.	Latency, Reliability, Computation Offloading

Liu, 2016 [15]	Edge computing minimizes bandwidth costs by shifting computational activities to edge devices, reducing cloud data transfer.	Edge devices and platforms can't communicate without established protocols and interfaces.	Task planning using the Markov decision process in MEC systems.	Results showed minimum delay compared to baseline approaches.	Average Delay, Task Scheduling.
Mao, 2016 [107]	Supporting real-time apps improves response times and user experiences.	Cloud or data center resources may be needed for computationally demanding processes or applications.	Using energy harvesting is a way to reduce the computational load on MEC.	The proposed model outperforms benchmarked policies.	Cost, Energy, Completion Time, Computation Offloading
Mao, 2017 [68]	Optimization minimizes network congestion, bandwidth costs, and efficiency.	Managing updates, maintenance, failures, and replacements in large-scale edge installations may be difficult.	Low complexity suboptimal method for MEC systems.	The results show less latency than standard processes.	Energy, Delay, Task Offloading, Resource Allocation

2.3 Related Work for SDN Integration with Resource Optimization Aspects

In this section, we discuss various resource optimization techniques in the SDN-integrated paradigm. The effects of SDN in edge computing are also discussed.

2.3.1 Resource Optimization for SDN Integrated Paradigm

SDN is a transformative technology that can revolutionize the performance of network-intensive services such as data centre management and network virtualization by centralizing control and management functions. This enhances flexibility and agility while improving the overall quality of service for users, rejuvenating the entire spectrum of services. Various resource optimization strategies have been proposed in the literature to use network resources and enhance system performance effectively. This paper examines the key advancements in SDN-based resource optimization schemes, offering a comprehensive overview of recent developments.

Rahman [111] presented an architecture that utilizes IoT networks and SDN support layers to enable efficient and intelligent industry management. The proposed architecture was evaluated through performance assessments on simulated environments with appropriate configurations.

To reduce network communication costs, Isyaku [112] proposed a flow monitoring strategy that combines node query requests and responses, utilizing heuristic-based optimization techniques to monitor the network. This approach helps to optimize resource allocation and reduce communication costs.

The author of [113] and her team researched attacks on actual IoT devices to emphasize the significance of security in IoT system design. The study revealed current security vulnerabilities in commercial IoT systems, stressing the need for essential security features such as reliability, anonymity, privacy, authorization, authentication, capability recovery, and self-organization.

Yurchenko [114] built a lightweight Microservices component for OpenNet VM to control network functions (NF) on demand. The potential for interoperability between NFs is made possible by OpenNet VM's support for high-performance. Their recommended NF handler fully leverages the service chain for packet management.

Maksymyuk [115] introduced an industrial 5G touch Internet solution based on SDN NFV. The system employs a layered architecture with cloud support and built-in intelligence to automate physical network functions in dynamically created, monitored, updated, and deleted Virtual Network Functions (VNFs). The aim is to enhance the calculation of IoT heterogeneity in industrial settings.

Antonakakis has described a virtualized SDN-compatible IoT architecture [116]. They took advantage of the edge by hosting SDN/NFV nodes that hosted the VNFs. The platform presented by the authors offers several benefits, such as low latency and the rapid deployment of IoT devices facilitated by edge devices, thanks to its ability to provide granular user context. The MANO plan is responsible for managing network and infrastructure operations while creating virtual gateways through NFV, which enables improved scalability, portability, and deployment speed. While the theoretical models presented in this article are promising, there is currently no evidence of their practical implementation or testing.

Du [117] discussed an IoT traffic management prototype that utilizes contextual forwarding and processing. Contextual data delivered from sensor and application layers in IoT networks can reduce scalability, security, reliability, and computational capabilities. Thus, software-defined data plane services are possible using programmable switches. The FLARE Platform IoT Gateway software offers contextual packet forwarding and processing, IoT device identification, and connectivity.

The authors [118] propose a novel technique that connects each cluster of patients using e-health IoT devices to an SDN edge node, based on their similar-

ity and distance from one another, to facilitate effective caching through spectral clustering. The suggested MFO-Edge Caching algorithm is considered for taking the near-optimal data alternatives for caching based on evaluated criteria to improve QoS. The proposed method has been demonstrated to enhance the cache hit rate and decrease the average data retrieval time by 76% in experimental settings. The strategy outperforms competing approaches, demonstrating the value of SDN-Edge caching and clustering in maximizing the efficiency of e-health networks.

The authors [119] presented a safe and energy-conscious design for fog computing, and we devised a load-balancing method to maximize efficiency in an SDN-enabled fog setting. A Deep Belief Network (DBN)-based intrusion detection system was developed to minimize delays in communication between tasks within the fog layer. Based on simulation results, it was determined that the suggested approach offers a successful technique for load balancing in a fog environment, with improvements of 15% in average reaction time, 23% in average energy usage, and 10% in communication latency compared to current methods.

The author [23] introduced a novel framework known as Software-defined Edge Cloudlet (SDEC), which is based on an RL optimization framework. This framework aims to address the problem of task offloading and resource allocation in wireless MEC. The recommended strategies for addressing the complex issue include using Q-learning and cooperative Q-learning-based reinforcement learning techniques. The simulation results indicate that the suggested approach effectively decreases the summation latency by 31.39% and 62.10% compared to alternative benchmark approaches, such as classic Q-learning with a random algorithm and Q-learning with epsilon greedy.

The author [120] proposed a load-balancing scheme for machine-to-machine (M2M) networks that considers traffic conditions. This scheme utilizes SDN technology. Load balancing techniques are crucial in M2M networks to handle the high demand caused by sudden traffic bursts. By using the power of SDN to

observe and manage the network, the suggested load balancing method can meet various quality of service needs by identifying and redirecting traffic.

The author [121] introduced a technology called Fibre-Wireless (FiWi) that can enhance VECNS (Vehicular Edge Computing Networks). FiWi technology offers advantages such as centralized network management and support for multiple communication techniques. The author developed a scheme to reduce the time vehicles take to process their tasks. The scheme utilizes SDN and load balancing to offload tasks. This is done in a FiWi-enhanced VECN environment, where SDN helps centrally manage network and vehicle information. Some recent optimization studies in which SDN integrates with edge computing and their findings, contributions, and results are discussed in Table 1.

Table 2.5: Related Work on Optimization in SDN

Author, Year & Reference	Findings/ Outcomes	Limitations	Research Contribution	Results	Optimized Parameters
Jazaeri, 2023, [118]	The centralized management and dynamic caching technique enhance the performance and reliability of IoT healthcare systems that communicate with patients directly.	If there are any sudden changes in network configurations and settings, IoT devices might have problems working together.	The MFO-Edge Caching algorithm is regarded as an approach for selecting near-optimal data alternatives for caching, utilizing evaluated criteria.	Improved quality of service and minimized the average data retrieval time.	Cache Rate, Delay-Time, Performance QoS, and Average time.

Jagdeep, 2022, [119]	A Deep Belief Network (DBN) reduces response time and average energy consumption through efficient load balancing.	Improve IoT integration while integrating with computing technologies like Edge and Fog to reduce workload. Security is also not considered.	An SDN-enabled fog environment is used to implement load balancing algorithms.	Minimized energy consumption and communication delay.	Energy, Execution time, Response Time, Average cost, Performance and Average Load
H.Liu, 2019, [23]	The RL and QL are also used to solve the delay minimization problems.	Security is a primary concern while the agents work on a multi-layered architecture.	A resource allocation and computational offloading scheme is proposed in MEC for SDN-based wireless networks.	Minimized delay, improved efficiency, and utilization of the system.	Total Delay and Efficiency
Zhang, 2019, [121]	The integration of SDN with FiWi technology is used to enhance vehicle computing-intensive tasks.	Lack of adaptability to real-world scenarios and scalability issues.	Better task offloading and load balancing improve the efficiency of the system.	Reduction in process delays by utilizing edge servers and improved resource utilization.	Process Delay, Running Time, and Average Utilization

Chen, 2018, [120]	The capability of SDN improves the system efficiency by monitoring and controlling the network.	The incorporation of Security parameters in such environments is still a challenge.	A load-balancing scheme is utilized for SDN-based M2M networks.	Improved QoS of the system and provided the benefits of load balancing.	Response Time, CPU workload, and resource utilization
-------------------	---	---	---	---	---

2.3.2 Resource Optimization Techniques in SDN-based Edge Computing

The authors proposed a novel approach named Hybrid Genetic, Max-Min, Min-Min, and Random Algorithms (GAMMR) [122] for determining the optimum mapping for independent tasks to a set of fog-cloud resources. Fog computing in IoT systems strives to reduce latency by moving processing closer to end users, and the suggested GAMMR algorithm outperforms in terms of balancing energy usage and response time.

In [123], the authors proposed a Two-stage Software-defined Task and Resource Allocation (TS-SDTRA) system for efficient task and resource management. This framework applies a Markov Decision Process (MDP) along with a Proximal Policy Optimization (PPO) algorithm to achieve an optimal balance between energy and load, all while following delay constraints. Two methods for setting rewards actively reduce energy consumption, balance workloads, and enhance server scheduling.

Furthermore, [124] proposed a resource allocation hybrid metaheuristic algorithm based on the strengths of the PSO and GWO called hybrid PSO-GWO. The resource allocation strategy aims to minimize energy consumption and response time. The research focuses on addressing the challenges in scheduling and concentrates on the limitations of existing resource allocation strategies that

affect the system cost and completion time of executed tasks. To overcome the issues in the existing system, the author proposed lightweight, time- and resource-efficient schemes designed to minimize application delays and costs. It introduces a mathematical model and adaptive strategies for handling tasks [125].

In [126], the author proposed a novel framework for selecting a slave controller in SDN-based IoT networking applications in a multi-domain scenario. The authors utilize a multi-criteria decision-making approach to analytically define the problem, which guarantees a strong and effective selection process. The strategy focuses on enhancing load-balancing abilities and notable advancements in multiple network performance metrics.

The [127] presented an SDN-based architecture for optimizing edge and cloud computing in 5G wireless HetNets. It proposed an evolutionary game for user service selection and a Stackelberg differential game for cloud resource sharing. The approach aims to maximize utilities for both the Central Computing Platform (CCP) and Edge Computing Providers (ECPs) while efficiently managing time-varying computational tasks from user devices. Simulation results validate the effectiveness of the proposed mechanism. Moreover, in [128], the author proposed a hybrid FDMA-NOMA-assisted dual computational job offloading technique for vehicle networks, along with an optimization approach to reduce delay and energy consumption costs. Additionally, the authors employed convex optimization and the DDPG algorithm to solve transformed subproblems, yielding results that outperform baseline schemes. A comparison of the technology used and parameters optimized in articles, focusing on key components such as load balancing, task scheduling, computational offloading, and resource allocation, is presented in Table.

Table 2.6: A comparison of existing techniques in SDN-based edge computing

Author, Year, and Refer- ence	Algorithm/ Tech- nique	Key Focus	Optimi- zation Method	Perfor- mance Metrics	Applica- tion Domain	Dis- advantages
---	------------------------------	--------------	-----------------------------	-----------------------------	----------------------------	--------------------

Tian, 2025, [129]	Task Offloading via Reinforcement Learning.	Reinforcement learning-based task offloading and allocation.	Deep deterministic policy gradient (DDPG) reinforcement learning.	Energy consumption, task delay, workload distribution	Vehicular networking and multi-access edge computing	Latency and energy trade-offs not fully optimized.
Zhu, 2025, [130]	NSGA-III	Offloading Task offloading in intelligent transport systems.	Multi-objective optimization using NSGA-III	Task delay, energy consumption	Intelligent transportation systems	High computational complexity for large-scale problems.
Darabkh, 2025, [131]	Velocity-Aware Routing	Routing in VANETs using clustering.	Weight function-based clustering and SDN	Throughput, end-to-end delay, control overhead	Vehicular ad hoc networks (VANETs)	Cluster stability is affected by vehicle speed variations.
Zhao, 2025, [132]	Platoon-Based Offloading	Resource allocation via platoon collaboration	Deep Reinforcement Learning and LSTM	Computation cost, latency, and offloading success rate	Vehicle edge computing	Requires accurate future route prediction for effectiveness.
Chai, 2025, [133]	Stackelberg Game-Based Allocation	Joint computing resource allocation and task offloading.	Bi-level optimization using Stackelberg game theory	Energy efficiency, task completion time	Edge computing	Bi-level optimization may lead to high computational overhead.

Wu.V, 2024, [134]	Mean-Field Game-Based Load Balancing	Load balancing for industrial mobile edge computing	Mean-field game theory and multi-agent reinforcement learning	Processing latency, load balancing efficiency	Industrial mobile edge computing	Load balancing inefficiency under dynamic workload conditions.
AL-Bakhrani, 2025, [135]	UAV-Assisted Multi-Objective Optimization	Dynamic resource allocation for UAV-assisted edge computing	Multi-objective reinforcement learning and adaptive control	Task completion rate, energy efficiency, and load balancing	UAV-assisted mobile edge computing	Resource allocation constraints impact system adaptability.
Jasim, 2024, [136]	Adaptive SDN-Based Load Balancing in Health-care	Load balancing in real-time health-care systems	SDN-based hybrid static and dynamic load balancing	System latency, deployment costs, workload distribution	Edge/fog computing in health-care systems	High dependency on the SDN controller for load balancing.

2.3.3 SDN and Network Functions Virtualization (NFV)

NFV is another important concept in the context of SDN. NFV is a technology that effectively complements SDN. It transforms network functionalities into software, making them easier to manage and utilize. By adopting this approach, it's possible to isolate network functions such as routing decisions from local devices and execute them on remote servers or in the cloud. SDN and NFV [137] share a symbiotic relationship, meaning they can benefit from one another without being dependent on each other for operation. Specifically, the virtualization and deployment of network functions can take place independently of SDN, and vice versa. NFV has the potential to offer numerous benefits, ranging from cost reduction to increased system openness.

In [138], the author introduced a new architecture that emphasizes flexible resource management and balanced resource utilization. This architecture integrates SDN and MEC to effectively allocate bandwidth and computing/storage resources. The SDN control modules within the cloud and MECS enable multiple wireless access networks to work cohesively and manage the significant volume of data being processed. The computing and storage capabilities at the MEC will help us meet various quality of service requirements. For instance, quick response times are crucial for safe and cooperative driving.

In this new architecture, various resource management methods are formulated as an optimization problem. This strategy aims to improve the utilization of computing power, storage, and bandwidth resources. The MEC controller uses SDN and NFV regulation modules to efficiently distribute bandwidth resources across multiple access technologies and base stations, thereby enhancing bandwidth utilization.

The future of communication networks is expected to depend on virtualized infrastructures that will provide customized services to meet dynamic traffic demands while also supporting heterogeneity and diversity. SDN and NFV [139] technologies offer significant benefits, such as reducing operational costs and enabling better resource utilization. They are gaining momentum and are anticipated to play a critical role in shaping the future of communication networks [140].

The authors [141] propose a resource consolidation scheme that leverages SDN control features to implement network resource management concepts. [142] presents a scalable NFV orchestration system that enables flexible distribution of cloud and broadband resources. Additionally, a security architecture based on SDN for 5G networks is proposed to address the underlying security challenge. The SDN-based MEC architecture effectively tackles the issue of selecting intelligent MEC servers and offloading data.

2.3.4 Resource Provisioning at Edge Computing using SDN

Edge computing is a viable paradigm for processing data closer to its source rather than sending it to centralized data centers or clouds. It is resource provisioning at the edge. The advantages of edge computing include increased reliability, reduced latency, enhanced privacy, and improved security.

Several factors must be considered for resource provisioning at the edge, including the location and characteristics of the devices, network connectivity, available bandwidth, and the requirements of the applications running on these devices. For example, devices with limited processing power or storage capacity may require offloading some of their computational tasks to nearby edge servers or clouds. In contrast, applications that require low latency or real-time processing may need to be run entirely at the edge.

There are several approaches to resource provisioning at the edge, including centralized, distributed, and hybrid management. In centralized management, resources are managed from a central location, such as a cloud or data center, and allocated to devices at the edge as needed. In distributed management, resources are managed locally on each device, with coordination between devices to ensure efficient utilization of resources. Hybrid management involves a combination of centralized and distributed management, with some resources managed centrally and others managed locally.

Several challenges are associated with resource provisioning at the edge, such as the dynamic nature of edge networks and the need to balance resource availability and energy efficiency. These challenges have led to the development of several resource provisioning frameworks and techniques, such as edge caching, load balancing, and resource prediction. Some notable research works on resource provisioning at the edge are also discussed.

Chen [143] introduced a framework for resource allocation and task scheduling in edge systems. This framework considers the energy efficiency and resource

availability of devices at the edge, ensuring they are utilized optimally.

Similarly, Zhang [144] introduced a deep reinforcement learning-based approach to resource allocation for edge computing to optimize the balance between performance and energy consumption. This approach is a remarkable innovation that has the potential to revolutionize resource allocation in edge computing.

Lastly, Sun [145] also made a significant contribution to the field of edge computing by introducing a load-balancing approach. Their approach considers the heterogeneity of devices and applications at the edge, aiming to maximize resource utilization while minimizing latency. These studies highlight the importance of effective resource provisioning at the edge and offer valuable insights into the challenges and opportunities in this area.

Resource management on the edge is a cutting-edge research realm that explores optimizing resource allocation and utilization in distributed edge computing environments. The author examines recent developments in this field using SDN [146].

Edge computing, a paradigm shift in computing, represents the technological prowess that enables data analysis and processing to be carried out close to the data source rather than restricted to a centralized data center. Nonetheless, managing resources in distributed edge environments can be a formidable undertaking, owing to the ever-evolving nature of edge devices and fluctuating workloads.

SDN is a revolutionary network framework that separates a network's control and data planes. This innovative approach allows network administrators to dynamically and centrally control network behaviour. SDN can address the obstacles associated with resource management in edge computing by providing a centralized control plane that effectively supervises the network and its resources [138]. This renders SDN the quintessential candidate for resource management on the edge. The following sections explain the dynamic resource management, resource allocation, and resource management aspects of SDN individually.

2.3.5 Dynamic Resource Management in Edge Computing using SDN

A recent research paper proposed a framework for dynamic resource management in edge computing using SDN [144]. The framework uses SDN to dynamically allocate resources to edge devices based on their workload and availability. The framework uses a reinforcement learning algorithm to optimize resource allocation based on workload and network conditions. The proposed framework comprises the SDN controller, edge devices, and a learning agent. The SDN controller manages the network and resources, while the edge devices perform the computation and storage tasks. The learning agent uses a Q-learning algorithm to learn the optimal resource allocation policy based on the current network conditions.

The proposed framework was evaluated using a simulation environment with multiple edge devices and varying workloads. The results showed that the framework could improve resource utilization by up to 60% compared to a static allocation policy. The framework could also adapt to changes in workload and network conditions, demonstrating its effectiveness in dynamic environments.

2.3.6 Resource Allocation in Multi-Access Edge Computing using SDN

Multi-Access Edge Computing (MAEC) is an emerging paradigm that enables mobile network operators to deploy compute and storage resources closer to end-users at the network's edge. A recent research paper [145] proposed an SDN-based resource allocation scheme for MAEC environments. The scheme employs a game-theoretic approach to allocate resources among multiple edge devices, ensuring both fairness and efficiency.

The proposed scheme comprises the SDN controller, edge devices, and a resource allocation module. The resource allocation module uses a Stackelberg game model to allocate resources among the edge devices. The model considers the utility and resource allocation cost of each edge device. The proposed scheme

was evaluated using a simulation environment with multiple edge devices and varying workloads. The results showed that the scheme could achieve higher resource utilization than a static allocation policy. The scheme could also ensure fairness among the edge devices, demonstrating its effectiveness in a multi-user environment.

2.3.7 Resource Management for Latency-sensitive Applications in Edge Computing using SDN

Latency-sensitive applications, such as those in augmented and virtual reality, require low latency to deliver a seamless user experience. The scheme utilises an ML algorithm to predict the latency of each application and allocate resources accordingly.

The proposed scheme comprises the SDN controller, edge devices, and a machine learning module. The ML module uses a Random Forest algorithm to predict the latency of each application based on the workload and network conditions. The SDN controller utilizes latency prediction to allocate resources to edge devices. The proposed scheme was evaluated using a simulation environment with multiple edge devices and varying workloads. Some of the proposed algorithms in SDN-based EC with their Optimization parameters and platforms are discussed in Table 2.7, and a Comparison between optimized parameters is shown in Figure 2.5.

Table 2.7: Summary of the Studies Proposed Algorithms in SDN-based Edge Computing

Author, Year & Reference	Proposed Algorithm	Key Concept	Optimized Parameter	Platform
Skondras, 2019, [147]	FLSA-CC QoS	Task Scheduling, Resource Allocation	Throughput, Waiting Time	Lte-Sim Simulator, iCan cloud, Omnet ++
Zhang, 2020, [148]	Online efficient energy approach	Task offloading, Task Scheduling	Energy, Running time	Realtime

Guo, 2016, [149]	SDN-based TCP Aggregation/scheduling smart grid framework	Aggregation and Scheduling	Throughput, Performance, Fairness	Smartgrid environment Scenario
Feng, 2014, [150]	Joint allocation model in SDN, Popular flow scheduling policy	Resource allocation and scheduling	Delay time, bandwidth, throughput	Mininet
A.Sharma, 2021, [151]	Priority based context information Pr-CAI algorithm	Task Scheduling and Resource scheduling	Delay time, Performance	Omnet ++, Sumo
Chen, 2022, [152]	Design and effective flow scheduling system	Task scheduling, Resource allocation	Cost, throughput, performance, complexity	Realtime

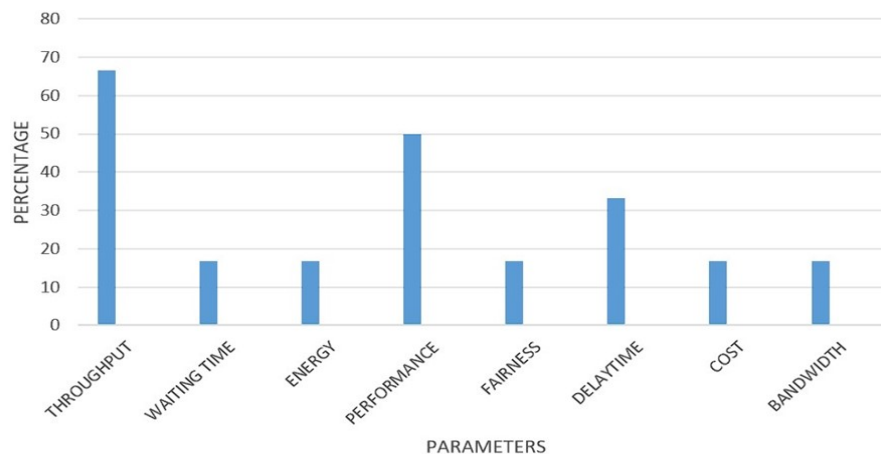


Figure 2.5: Optimized Parameters

2.4 Open Issues, Challenges, and Current Trends for Integration of SDN-based Edge and IoT

Several unresolved problems, difficulties, and emerging trends exist in integrating SDN-based edge paradigms with IoT [147]. Some essential factors to remember in this area are discussed below:

- I. Scalability and Heterogeneity:** IoT edge ecosystems comprise numerous heterogeneous IoT devices, each with distinct capabilities, protocols, and connection requirements [148]. Scalability and managing heterogeneous devices are issues when integrating SDN with Edge-IoT-CC. A significant issue remains in ensuring seamless interoperability and effective resource management across a wide and varied ecosystem [153].
- II. Security and privacy:** Due to the proliferation of linked devices, security and confidentiality are of the utmost importance in the CoT (Integration of IoT and CC). To secure sensitive [154] data, SDN-based integration must consider authentication, access control, data integrity, and encryption concerns. The vast volumes of personal data generated by IoT devices also raise significant privacy concerns, necessitating the careful design and deployment of privacy-preserving methods.
- III. Network Latency and Dependability:** Applications often require severe network latency and dependability [155]. The dynamic nature of IoT devices, changing network circumstances, and the need for in-the-moment data processing must all be considered when integrating SDN with CoT. In a dynamic CoT context, achieving low-latency communication, dependable connections, and effective routing techniques remains challenging.
- IV. Resource Management and Orchestration:** It is crucial to manage and coordinate resources effectively in a CoT ecosystem. SDN may provide centralized management and programmability, but controlling traffic flows, distributing network [156] and computing resources to IoT devices, and optimizing resource use are continuous issues. Critical issues must be addressed, including load balancing, QoS assurances, and dynamic resource allocation.
- V. Interoperability Issues:** These issues may arise because CoT and the IoT use various communication technologies, standards, and protocols. The devices' heterogeneity must be addressed to ensure smooth communication across multiple platforms, networks, and applications. Standardization ac-

tivities are necessary to provide standard protocols, interfaces, and data models for effective integration. The following are current developments in CoT and IoT integration using SDN.

- VI. **Edge and Fog Networking:** When combining SDN with CoT, Edge and fog networking paradigms are often used to overcome latency and reliability issues. By moving processing power closer to the network's edge, data transmission can be minimized, reaction times sped up, and performance enhanced [157].
- VII. **Machine Learning and AI-Driven Management:** Machine learning and artificial intelligence are becoming increasingly popular methods for managing and optimizing. SDN-based edge networks. To cope with the dynamic nature of IoT settings, these technologies may enable proactive resource management, predictive analytics, and automated decision-making.
- VIII. **Edge Intelligence and Analytics:** Conducting data analytics and intelligence at the network's edge is crucial. SDN-based CoT systems may extract valuable insights in real-time, minimize data transfer to the cloud, and improve privacy and security by processing data locally on edge devices. Blockchain technology is being investigated to enhance security, trust, and data integrity in CoT and IoT contexts. SDN-based CoT systems may provide decentralized authentication, secure data exchange, and transparent transactional records using blockchain-based solutions.
- IX. **5G and Network Slicing:** Introducing 5G networks and network slicing tools [158] opens up many possibilities for SDN-based CoT and IoT integration. The performance and flexibility of IoT applications are enhanced by 5G's increased bandwidth, ultra-low latency, and network slicing, which enable the construction of virtualized networks tailored to their unique requirements.

The practical and safe integration of SDN-based CoT and IoT ecosystems will be

significantly aided by addressing these open concerns and difficulties while using existing trends.

2.5 Contextual Summary

SDN is a promising extension of EC due to its specific management attributes and solutions for the needs of emerging technologies. The present study analyzed some new state-of-the-art findings in Edge Computing, SDN, and their integration. We have classified the objectives of resource optimization in edge computing into four key areas: load balancing, computation offloading, resource allocation, and task scheduling. Additional categorizations are being considered for SDN's function in optimizing edge computing. The considered methodology defined the magnitude of the problem formulation, particularly for the scenario and underlying assumptions. This chapter analyzes the discussion and research problems, followed by a proposal of three recommendations. Firstly, it is necessary to consider the gains, limitations, and optimization of resources in both paradigms to effectively employ the end devices. Furthermore, the issue of resource orchestration among multiple providers can be effectively addressed by implementing a multiparadigm system that combines SDN with NFV. Lastly, it is crucial to incorporate dynamism into the scenario to replicate real-time computing system situations. It has also been observed that SDN-based technology can effectively resolve various unresolved matters related to effective resource allocation in MEC.

Chapter 3

Hybrid PSO-GWO for Resource Allocation in SDN-based Edge Computing

The demand for an effective controller to guarantee optimal resource utilisation in edge computing has increased due to the exponential growth of the Internet use and the dispersed nature of computing resources in edge devices. In this study, we focus on one particular aspect of edge computing: the issue of allocating resources, to minimize time delays while conserving the battery power of user devices. For this purpose, we introduce a novel hybrid metaheuristic specifically designed to optimize total time and energy consumption in a hybrid SDN based Edge-Cloud computing environment. The proposed algorithm leverages the strengths of both PSO and GWO techniques. To evaluate the effectiveness of our approach, we conducted a comprehensive comparison between the Hybrid PSO-GWO, PSO, and GWO methods. Through simulations and experiments, we examined their performance in terms of time delays and energy consumption. The results demonstrate that the Hybrid PSO-GWO algorithm outperforms both PSO and GWO, proving its efficacy in achieving more efficient and effective resource allocation in wireless edge computing scenarios.

3.1 Proposed Model

In this section, we are discussing the system architecture and its working mechanism at different layers. Besides, we discuss problem formulation and proposed mathematical formulation for the approach. At last, we presented and discussed

the proposed algorithm

3.1.1 System Architecture

Figure 3.1 depicts the three tiers that make up the planned network architecture: the SDN control layer, the edge computing layer, and the infrastructure layer. An SDN controller, part of the SDN control section, provides open unified protocols for managing tasks, resources, and user accessibility in a programmatic manner. The SDN controller may retain data on network traffic and the energy characteristics of edge servers since it has a comprehensive picture of the network architecture, allowing for dynamic, network-wide control. The SDN controller can leverage its comprehensive knowledge of the network to achieve optimal performance.

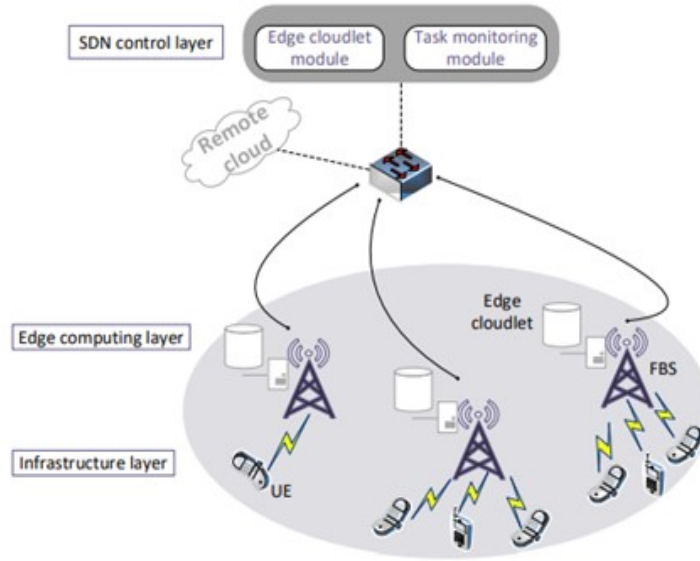


Figure 3.1: Architecture of SDN-based Networks

The Task Monitoring Module in the SDN control layer collects user equipment (UE) task data like computation. It advises on whether to run tasks on edge nodes or clouds or in the UE's local computing environment. The Edge Cloudlet Module tracks distributed edge server memory, CPU, and load. This data aids task offloading and resource allocation. Edge computing includes virtual edge cloudlets (ECs) and femto-base stations (FBSs). FBSs and ECs form edge nodes (ENs). ENs relieve network nerve centres by mediating UE-edge cloudlet com-

munication. They forwards network data and provide limited control. ENs view wireless and core networks holistically. Deploying ENs in SDN allows seamless integration of wired and wireless aspects, reducing controller load.

ECs, on the other hand, are virtualized servers located near FBSs. They provide computing functionalities to users and serve as an alternative to accessing cloud data centers. By offloading tasks to ECs, users can avoid the energy consumption, latency, and cost associated with accessing remote cloud servers. OpenFlow enabled switches facilitate communication between ECs and the SDN controller. Incorporating cloud services, ECs control all aspects of computation and storage, allowing for cutting-edge methods of traffic management including offloading and caching.

The infrastructure layer comprises physical network entities responsible for task computation, including UEs. In this layer, a scenario is considered where FBSs are presented as $N = \{n_1, n_2, n_3, \dots, n_i\}$ are deployed, each equipped with an ECs, to serve j user's tasks represented by $J = \{j_1, j_2, j_3, \dots, j_j\}$. Each ECs has a finite computation capability and is located at the edge of the radio access network to provide computing services and execute UE requests. The required computing task size for each UE and a remote cloud (RC) is also present to offload overloaded requests from ECs for further execution

3.1.2 Problem Formulation

According to our architecture, the flow of various tasks and other processes are expressed in terms of mathematical equations in different scenarios.

3.1.2.1 Local Computing

At UE_i we are thinking about Local computing with c_i computing capability. For each bit, the CPU spends ϵ_i represents number cycles per bit, s_i is the size of computation tasks and R_i represents the local task ratio. Thus, task's local computation time, total requested tasks and consumed energy $t_{s_i}^{UE}$, T_{rt} and $e_{c_i}^{UE}$ respectively are can be calculated as:

$$t_{s_i}^{UE} = R_i s_i \frac{\epsilon_i}{c_i^{UE}}, \quad T_{rt} = \sum_{i=1}^M s_i, \quad e_{c_i}^{UE} = (K_i \epsilon_i) R_i s_i \quad (3.1.1)$$

Where; K_i is the power coefficient of energy consumed per CPU cycle for local computation.

3.1.2.2 Transmission Time and Energy Consumption

The literature specifies the transmission time of user equipment UE_i and amount of energy that may be saved for data offloading between UE_i and FBS_j are $t_{i,j}^{OL}$ and $e_{i,j}^{OL}$ respectively can be calculated as:

$$t_{i,j}^{OL} = (1 - R_i) \frac{s_i}{r_{i,j}}, \quad e_{i,j}^{OL} = \frac{p_i(1 - R_i)s_i}{r_{i,j}} \quad (3.1.2)$$

3.1.2.3 Execution Time in Edge Computing

The edge cloudlet will carry out UE requests once it has received them. As the edge cloudlet has restricted processing capabilities (CPU cycles per second), it is assigned to user i by the j th FBS as $l_{i,j}^t$. The amount of time it takes for UE_i to send data in the j^{th} edge cloudlet is

$$t_{i,j}^{tr} = t_{i,j}^{OL} + t_i^{ex} = (1 - R_i) \frac{s_i}{r_{i,j}} + \omega_i s_i \frac{\epsilon_i}{c_{i,j}^c} \quad (3.1.3)$$

3.1.2.4 Sending Overloaded Requests to Remote Cloud

Since the RC can handle all UE tasks, it must be flooded with more task requests. Transmission to RC takes a while as

$$t_j^{ttr} = (1 - R_i - \omega_i) \frac{s_i}{r_j} \quad (3.1.4)$$

Here r_j is FBS_j 's data rate to the RC. Wire-based fronthaul lines connect FBS's and RC's. Wired connections are faster and more reliable. UE_i takes the total time of complete UE_i

$$\begin{aligned} t_{ex}^{full} &= t_{s_i}^{UE} + t_{i,j}^{tr} + t_j^{ttr} \\ t_{ex}^{full} &= R_i s_i \frac{\epsilon_i}{c_i^{UE}} + (1 - R_i) \frac{s_i}{r_{i,j}} + \omega_i s_i \frac{\epsilon_i}{c_{i,j}^c} + (1 - R_i - \omega_i) \frac{s_i}{r_j} \end{aligned}$$

3.1.3 Energy Formulation

In light of the previous analytical findings, our objective is to cut down as much as possible on the overall amount of time needed for execution while adhering to the

energy consumption. The issue may be expressed using a mathematical equation as follows:

$$E = \sum_{n=1}^N \sum_{m=1}^M E_n \cdot X_{nm} \quad (3.1.5)$$

$$\sum_{n=1}^N X_{nm} = 1; \text{ for } m = 1, 2, \dots, M \quad (3.1.6)$$

$$P_n \sum_{m=1}^M X_{nm} \geq C_m; \text{ for } n = 1, 2, \dots, N \quad (3.1.7)$$

$$B_n \sum_{m=1}^M X_{nm} \geq L_m; \text{ for } n = 1, 2, \dots, N \quad (3.1.8)$$

$$X_{nm} \in \{0, 1\} \quad (3.1.9)$$

$N \rightarrow$ Total number of edge devices.

$M \rightarrow$ Number of computational tasks or services.

$P_n \rightarrow$ Processing capacity of edge device n .

$B_n \rightarrow$ bandwidth of edge device n .

$C_m \rightarrow$ Computational requirement of task m .

$L_m \rightarrow$ Latency tolerance of task m .

$E_n \rightarrow$ Energy consumption of edge device n .

$X_{nm} \rightarrow$ Binary decision variable indicating whether task m is allocated to edge device n (1 if allocated, 0 otherwise)

In this formulation, the objective is to minimize the total energy consumption of the allocated tasks across edge devices. The constraints ensure that each task is allocated to exactly one edge device, the processing capacity of each device is sufficient for the allocated tasks, and the available bandwidth meets the latency requirements.

$$Fitness = w * t_{ex}^{full} + (1 - w) * E \quad (3.1.10)$$

The PSO-GWO hybrid optimization algorithm is applied to find the optimal binary allocation variables X_{nm} that satisfy the objective and constraints. PSO is employed to explore the solution space efficiently, while GWO is used to enhance the global search capability and convergence speed.

3.1.4 The Algorithm

The algorithm employs a hybrid framework that combines the principles of PSO and GWO to efficiently determine optimal task assignments onto specific edges

integrated with SDN architecture. This process is guided by a multi-objective fitness evaluation that considers energy consumption, task latency, and resource utilization. The algorithm's fundamental steps encompass a sequential process like initialization of variables, calculation of matrices, evaluation of metrics, weight assignment, fitness computation and return respectively. The flow of algorithm in steps is shown in figure 2. PSO is a basic bio-inspired algorithm for searching

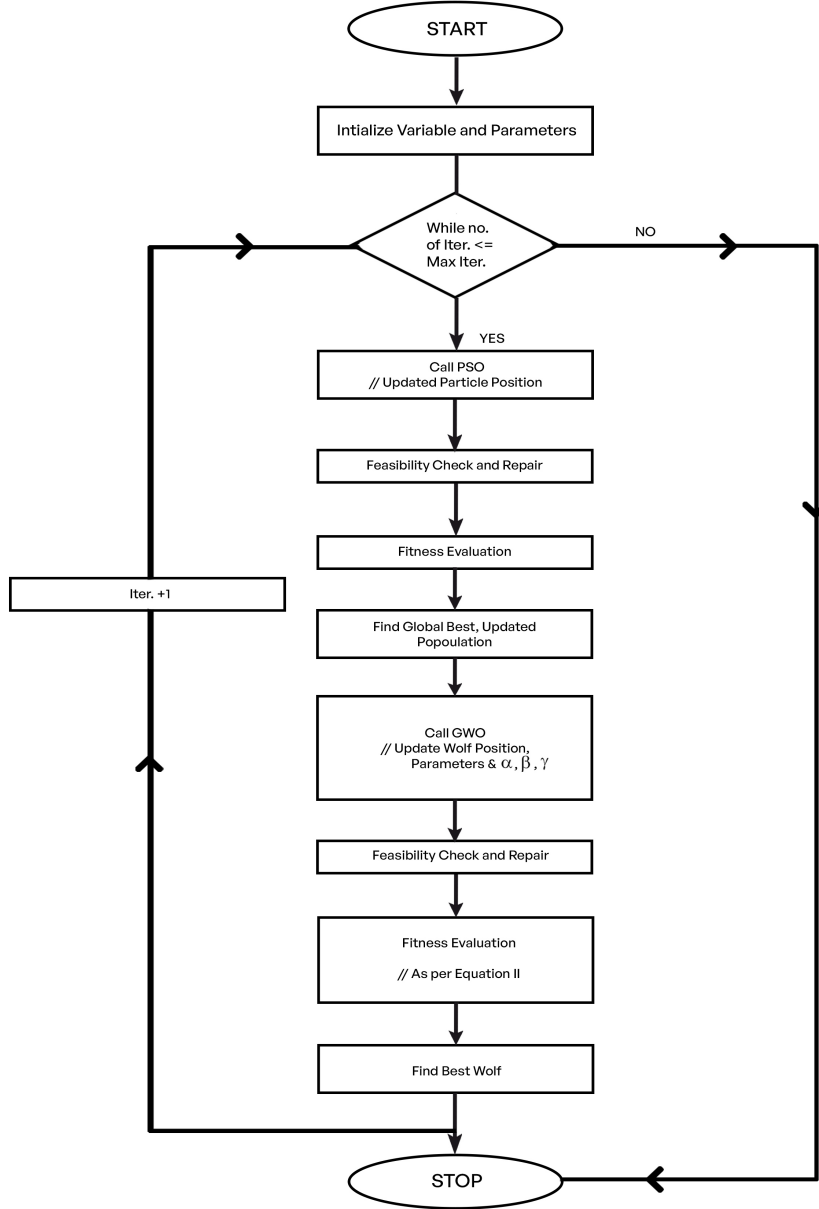


Figure 3.2: Flow chart of Proposed Algorithm

the solution space for the best solution. It simply needs the performance index and is unaffected by the contour or any differential form. To mimic the complex social hierarchy and hunting strategy of wild grey wolves, the GWO algorithm

is devised. Four types of grey wolves, called alpha, beta, delta, and omega, are employed to act out a leadership hierarchy.

Algorithm 1 Hybrid PSO-GWO Algorithm

```

1: procedure INITIALIZE PSO AND GWO POPULATIONS RANDOMLY:
2:   -PSO population: N_particles
3:   -GWO population: N_wolves
4:   Initialize velocity  $V_i$  and position  $X_i$ 
5:   Initialize parameters:  $w, c_1, c_2$ 
6:   Evaluate initial fitness:
7:    $Fitness = w * t_{ex}^{full} + (1 - w) * E$ 
8:   Set  $Pbest_i = X_i$  and determine  $Gbest$ 
9:   Identify  $\alpha, \beta, \delta$  wolves
10:  Repeat for a Maximum of max_iterations or until termination criteria are
    met:
11:    for Each PSO Particle in Population: do
12:      Update velocity:
13:       $V_i = w * V_i + c_1 * r_1 * (Pbest_i - X_i)$ 
14:       $+ c_2 * r_2 * (Gbest - X_i)$ 
15:      Update position:
16:       $X_i = X_i + V_i$ 
17:      Apply constraint:  $X_{nm} \in \{0, 1\}$ 
18:      Evaluate fitness of particle
19:      Update  $Pbest_i$  and  $Gbest$ 
20:    end for
21:    for Each GWO Wolf in Population: do
22:      Update position using GWO
23:      Apply constraint:  $X_{nm} \in \{0, 1\}$ 
24:      Evaluate fitness of wolf
25:    end for
26:    Update  $\alpha, \beta, \delta$ 
27: end procedure

```

In this algorithm we use both algorithms one by one. After initialization we call the PSO and updated the particle position and find the global best and on updated position we call GWO and after feasibility check and fitness evaluation and get our best wolf.

3.2 Experimental Setup and Result

For the evaluation of delay time and energy consumption, we conducted our study using MATLAB 2020a. The utilized simulation parameters and experimental setup conditions are discussed in Table 1. The main objective of this research was

to compare three different optimization methods: PSO, GWO and our proposed hybrid PSO-GWO.

Table 3.1: Simulation Parameters

Parameters	Amount
FBSs Counted	5
UEs per FBS Counted	10
Increased Channel power	$127+30.\log$ (d)
power of UE, p_i	0.5 W
UE bandwidth for data transmission, B	10 MHz
Total battery capacity, $E_i^{UE.max}$	500 J
Error bars for Gaussian channel, σ^2	3×10^{-15} W
Number of Iteration	100
population size	50
inertia weight(w)	0.9
cognitive coefficient(c1)	1.4
social coefficient(c2)	1.4
lb and ub	-10 and 10
No. of leader	3

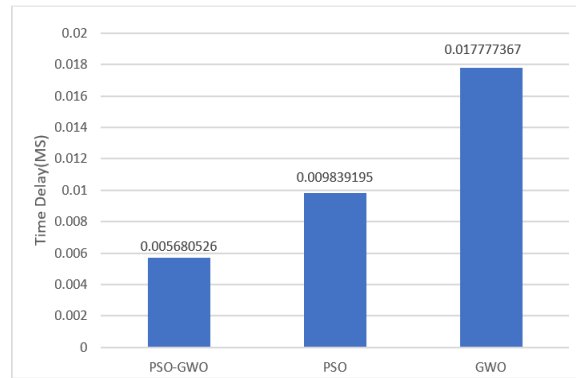


Figure 3.3: Time Delay Comparison

The plot presented in figure 3 illustrates the time delay evaluation for three different methods: PSO, GWO, and the proposed PSO-GWO. The results indicate that the PSO-GWO method exhibits the lowest time delay among the three methods, with a value of 0.00568. In comparison, the PSO method has a time delay of 0.00983, and the GWO method has the highest time delay of 0.017777.

These findings demonstrate the superior time efficiency of the PSO-GWO approach compared to both PSO and GWO. The PSO-GWO method's ability to achieve significantly faster execution times makes it a more efficient option for solving optimization problems, presenting promising potential for real-world applications where time-critical solutions are required.

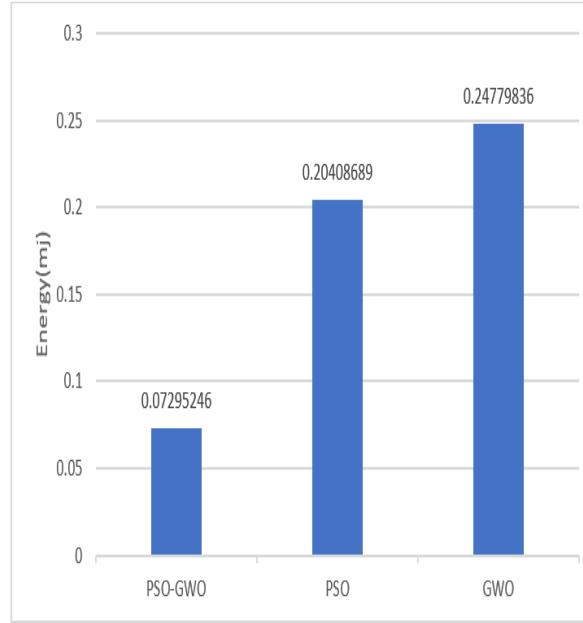


Figure 3.4: Energy Optimization Comparison

The figure 4 plot depicts the energy consumption evaluation for three different optimization methods: PSO, GWO, and the proposed PSO-GWO. The results indicate that the PSO-GWO method consumes the least amount of energy among the three methods, with a value of 0.00729. In comparison, the PSO method has an energy consumption of 0.020408, and the GWO method exhibits the highest energy consumption of 0.02477. These findings highlight the superior energy efficiency of the PSO-GWO approach compared to both PSO and GWO. The PSO-GWO method's ability to significantly reduce energy consumption makes it a more environmentally friendly and cost-effective option for solving optimization problems. This demonstrates its potential as a practical and sustainable solution for real-world applications where energy efficiency is a critical concern

3.3 Contextual Summary

The present study focused on evaluating the performance of three optimization methods, namely PSO, GWO, and the proposed PSO-GWO, in terms of time delay and energy consumption. The results clearly demonstrate the superiority of the PSO-GWO approach in both aspects. Regarding time delay, the PSO-GWO method outperformed both PSO and GWO, achieving the lowest time delay of 0.00568. This superior time efficiency makes the PSO-GWO approach a highly attractive option for solving time-critical optimization problems, offering promising potential for real-world applications. In terms of energy consumption, the PSO-GWO method proved to be the most efficient, consuming the least amount of energy with a value of 0.00729. This eco-friendly and cost-effective characteristic makes the PSO-GWO method an ideal choice for solving optimization problems with a focus on conserving energy resources.

Chapter 4

An Average based Resource Allocation and Load Balancing Strategy in SDN-based Edge Computing

Dynamic and distributed systems for modern task execution require effective strategies to ensure optimal performance, especially when managing various workloads with different timelines. In SDN-based edge computing, the tasks come from heterogeneous resources that have different task sizes and are highly dynamic. Therefore, an optimized resource allocation and load balancing scheme is required for the execution of all tasks. The main goal is to reduce overall task completion time, makespan, improve average resource utilization, and maintain even load distribution across edge servers to prevent bottlenecks and underutilization.

To address common challenges in such environments, this chapter introduces a task scheduling, resource allocation, and load balancing strategy called Average-Based Resource Allocation and Load Balancing (ABRL). The ABRL model is designed to provide fair and efficient task-to-server mapping that optimises key QoS parameters. Unlike traditional algorithms, the ABRL method focuses on distributing tasks among servers to prevent both over- and under-utilisation by balancing functions according to the mean task size and continuously monitoring each server's status.

The ABRL is proposed for efficiently allocating Edge servers and adequate load distribution across them. The algorithm is divided into three phases: Task Rearrangement, Resource Allocation, and Load Balancing. In task rearrangement, the tasks are collected in a queue and further rearranged in another queue by

calculating the average weight of all tasks. The task having task sizes near the average weight size will be selected in the rearranged queue. Afterwards, the tasks are assigned to the resources that have the minimum completion time. Thereafter, the adequate load distribution phase will start. The load balancing is done based on the completion time of servers. The migration of tasks from one server to another is also done in this phase. The key contributions of the paper include:

- ABRL algorithm is proposed for efficiently allocating edge computing servers and adequate load distribution across them.
- A three-layer SDN-based edge architecture is designed to implement the algorithm that gains the benefit of SDN in edge computing to improve the system's performance.
- A server allocation and load balancing technique is developed, enabling optimal resource utilization of computational resources to satisfy the demands of edge computing environments.
- Tasks are allocated through a strategic approach aimed to optimize the completion time, resource utilization, and level of load balancing in an SDN-based edge computing environment, thereby enhancing the overall efficiency and effectiveness of the system.
- The performance evaluation has been conducted using the ODL controller and implemented in Python.
- The proposed algorithm is compared with the existing benchmark heuristics like First-Come-First-Serve (FCFS) [66], Shortest Job First (SJF) [67], Dynamic Weighted Edge Computing Scheduler(DWECS) [69], Max-Min, Min-Min [72], Parallelization Task Allocation and Load-balancing algorithm (PTAL) [70], and Assigning Computing Tasks among multiple Edge Computing Servers (ACTE) [71] algorithms.

The DWECS is proposed for scheduling dynamic computational tasks among edge servers by integrating FCFS, SJF, and EDF (Earliest Deadline First)[68]. However, this algorithm is more suitable for task execution only and lacks its role in the server selection mechanism. This has a significant impact on overall system performance. In a similar chain, SJF is proposed for performing task scheduling while prioritizing the shortest task. This prioritization leads to starving the larger-sized tasks and thereby degrades several QoS metrics such as overall

completion time and makespan. Apart from these algorithms, FCFS is also proposed for performing resource allocation. However, this algorithm suffers from several disadvantages by not considering the server load conditions. PTAL uses effective task allocation and performs efficient load balancing, however, it suffers from starvation of large-sized tasks by not considering task size while performing task allocation and balancing the load. Likewise, ACTE was proposed for task assignment but lacks significant consideration related to task size and focus on the dynamic foundation. Further, this algorithm does not consider any strategy for balancing the load across servers. Apart from this, Min–Min and Max–Min were also proposed for efficient resource utilization, where Max–Min performs optimally and optimizes the resource utilization. Besides, Min–Min was found to be less significant as it schedules tasks by being more conservative towards the smallest task size. The existing study primarily focused on dynamic limitations for balancing the load among servers in a dynamic environment. However, the load balancing based on task completion time across servers is still unexplored and has great potential for optimization.

4.1 System Design

The proposed system is designed to support resource allocation and load balancing in SDN-based edge computing. This section discussed the proposed system architecture, problem formulation, and algorithm with an illustrative example.

4.1.1 The System Architecture

In this section, the SDN-based edge computing paradigm is demonstrated for the implementation of the proposed ABRL. This paradigm for ABRL is demonstrated in Figure 4.1. A multi-layered architecture that contains an infrastructure layer, a control layer, and an application layer is proposed and discussed below:

Application Layer: It is the top layer of the proposed architecture. In this layer, several users send multiple tasks. This layer is a key gateway for users to interact with the edge computing environment. Users submit their task execution requests to this layer. These requests are forwarded to the control layer, where further processing and resource allocation take place.

Control Layer: The system’s central layer is directly connected to the applica-

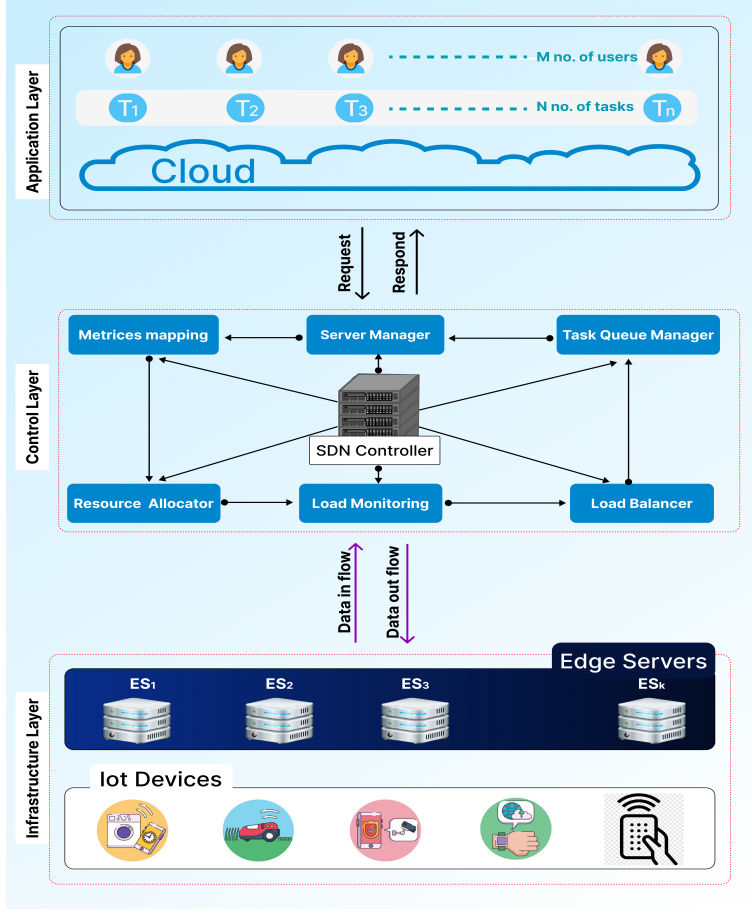


Figure 4.1: Proposed Architecture for SDN-based Edge Computing

tion layer and infrastructure layer with northbound and southbound Application Programming Interfaces (API). It helps to communicate between the user, tasks, and server and works as a middleware. It contains an SDN controller that acts as the brain of the system. It controls the resources for effective task execution to improve service quality and improve system efficiency. The controller provides smart and flexible resource management and load balancing. This layer is responsible for data management, system flow control, and better communication between all layers and users.

SDN Controller: The SDN controller is the core component of our system, managing the flow of data and control signals between the application layer and the infrastructure layer. Specifically, we use the OpenDayLight (ODL) controller, a widely adopted open-source SDN controller. It simplifies the network and edge resources, giving a clear and complete view that makes it easier to efficiently manage and coordinate tasks and resources. The ODL controller manages resources dynamically, enhancing Quality of Service and overall system efficiency.

The controller contains various components that demonstrate the operations of the proposed system, which are discussed below:

- i Task Queue Manager: It stores information about tasks and their size. Since the edge environment is dynamic therefore storing tasks information on their arrival is complex. This component collects the tasks at a point in time and makes a batch of them. Afterward, re-arranges them into another queue according to the proposed algorithm discussed in the Section 4.1.3.
- ii Server Manager: It is the essential component of the control layer. It helps to communicate with the application layer and has the necessary information about servers in the system, such as load, computation capability, and the location where they are deployed. It also helps in data management, monitoring, and task assignment to resources.
- iii Matrices Mapping: It is responsible for monitoring various parameters that are important for minimizing the task completion time and enhancing average resource utilization. It helps to maintain the information stored at the Server Manager and Task Queue Manager. By utilizing this information, it will do the best possible mapping with the task size and computation capability of servers as defined in Section 4.2. Thereafter, provide this information to the resource allocator.
- iv Resource Allocator: This component allocates the resources to the tasks according to the mapping provided by the mapping component. The task after rearrangement will be stored in a new/rearranged queue by the task queue manager.
- v Load Monitoring: This component is responsible for monitoring the load on every server and passing the information to the controller. It will help in deciding the load allocation across resources. The load information is further submitted to the load balancer.
- vi Load Balancer: It reallocates the tasks based on the information received from the above component. It works in a manner so that no server within the environment is either overloaded or underloaded. The aim is to achieve adequate load distribution across all servers so that maximum resource utilization can be obtained.

Infrastructure Layer: This layer is present at the southbound of the SDN control layer. It is a combination of the edge layer, or data forwarding layer, and the physical layer, or data creating layer. The main responsibility of the infrastructure layer is to manage the physical and logical resources essential for the edge computing environment. It involves a variety of objects, sensors, and systems that capture and process data at the periphery, as well as network infrastructure for connecting these devices to the wider computational environment.

The edge layer plays a vital role in processing data locally by placing data sources near the network's edge. It helps to make faster communication and reduce latency. The data is generated from the physical layer to give the response to the user's request from the application layer. The set of edge servers is deployed on the edge layer to execute the task and is directly controlled by a centralized SDN controller. The infrastructure layer is directly controlled by the control layer of the system. The aim is to design this environment to enhance resource utilization in the edge environment.

4.1.2 Problem Formulation

In the proposed system, the set of tasks denoted by $T_0 = \{t_i^0 : i = 1 \text{ to } N\}$. The set of edge servers $ES = \{ES_k : 1 \leq k \leq K\}$ having computing/processing capacity of $C[ES_k]$ in Million Instructions Per Second (MIPS).

Resource Allocation is a critical aspect of edge computing. An efficient resource allocation benefits the system by optimizing the QoS parameters. All the edge servers are directly controlled by the SDN controller. The information about servers, like their computation capacity, load upon them, and execution time for each task, is stored at the control layer.

- I. **Task Rearrangement:** After the task arrives from the task set T_0 , the task will be stored in the task queue. Afterward, the task will be rearranged into the rearranged task queue, i.e., T , by selecting the most nearby task size of the average size of all tasks. Let T denote the batch of newly arranged tasks as $\{t_i : i = 1 \text{ to } N\}$. The process of rearranging the task queue is formulated by calculating their average weight (ω).

$$\omega = \frac{\sum size[t_i]}{N} \quad (4.1.1)$$

Now, we will search for the task t_1 which minimize the difference $|size[t_i^0] - \omega|$, for $i = 1 \text{ to } N$. Then, find t_2 that minimize the difference $|size[t_i^0] -$

$size[t_1]|$, for $i = 1$ to N , where $t_i^0 \neq t_1$. Repeating the proposed procedure, we will rearrange the task into another queue, i.e., T. An optimized resource allocation Mapping (M) for a set of independent Tasks (T) that are forwarded for execution to the corresponding Edge Server (ES) denoted as:

$$M = T \rightarrow ES \quad (4.1.2)$$

The process of task rearrangement is done at the control layer of the system and flow is controlled by controller.

II. Resource Allocation: Afterward, we will calculate the time taken by all tasks on each server. This is derived by dividing the task size $size[t_i]$ by computation capacity of an edge server $C[ES_k]$.

$$\begin{aligned} ET &= [ET(t_i, ES_k)]_{N \times K} = \left[\frac{size[t_i]}{C[ES_k]} \right]_{N \times K} \\ &= \begin{bmatrix} \frac{size[t_1]}{C[ES_1]} & \frac{size[t_1]}{C[ES_2]} & \cdots & \frac{size[t_1]}{C[ES_{K-1}]} & \frac{size[t_1]}{C[ES_K]} \\ \frac{size[t_2]}{C[ES_1]} & \frac{size[t_2]}{C[ES_2]} & \cdots & \frac{size[t_2]}{C[ES_{K-1}]} & \frac{size[t_2]}{C[ES_K]} \\ \frac{size[t_3]}{C[ES_1]} & \frac{size[t_3]}{C[ES_2]} & \cdots & \frac{size[t_3]}{C[ES_{K-1}]} & \frac{size[t_3]}{C[ES_K]} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{size[t_N]}{C[ES_1]} & \frac{size[t_N]}{C[ES_2]} & \cdots & \frac{size[t_N]}{C[ES_{K-1}]} & \frac{size[t_N]}{C[ES_K]} \end{bmatrix} \end{aligned} \quad (4.1.3)$$

The total execution time of all tasks assigned to edge servers ES_k determines the edge servers' cumulative load, which can be stated as follows:

$$\begin{aligned} [CL[ES_k]]_{1 \times K} &= RT[ES_k] + \left[\sum_{\forall t_i \rightarrow ES_k} ET(t_i, ES_k) \right] \\ &= \begin{bmatrix} RT(ES_1) \\ RT(ES_2) \\ RT(ES_3) \\ \vdots \\ RT(ES_K) \end{bmatrix} + \begin{bmatrix} \sum_{\forall t_i \rightarrow ES_1} ET[t_i, ES_1] \\ \sum_{\forall t_i \rightarrow ES_2} ET[t_i, ES_2] \\ \sum_{\forall t_i \rightarrow ES_3} ET[t_i, ES_3] \\ \vdots \\ \sum_{\forall t_i \rightarrow ES_K} ET[t_i, ES_K] \end{bmatrix} \end{aligned} \quad (4.1.4)$$

On the other hand, the execution time of previously assigned tasks on ES_k will be used to estimate the Ready Time $RT[ES_k]$. The tasks T are allocated to ES_k that have min $CL[ES_k]$.

The resource allocation is done at the infrastructure layer and controlled by the middle layer implemented with an Openflow protocol.

III. Load Balancing: After the allocation of tasks, we have to identify maximum-loaded ES_k and minimum-loaded ES_k according to their $CL[ES_k]$. maxloaded

ES_k and minloaded ES_k is the server that has a maximum value of $CL[ES_k]$ and a minimum value of $CL[ES_k]$

$$\text{maxload } ES_k = \max CL[ES_k] \quad (4.1.5)$$

$$\text{minload } ES_k = \min CL[ES_k] \quad (4.1.6)$$

Now migrate the load from maxloaded ES_k to minloaded ES_k by calculating their difference(diff) as given below.

$$\text{diff} = \max CL[ES_k] - \min CL[ES_k] \quad (4.1.7)$$

Migrate the tasks that have $\text{size}[t_i] \sim x \text{ or } y$ from maxloaded ES_k to minloaded ES_k . The total migrated load in the server is considered as $\Delta CL[ES_k]$.

$$x = \text{diff}/2 \quad (4.1.8)$$

$$y = \text{diff}/4 \quad (4.1.9)$$

$$\Delta CL[ES_k] = \text{size}[t_i]/(C[ES_k]) \quad (4.1.10)$$

After migration, balance the execution time according to the migrated tasks t_i to the respective ES_k . Further, update the CL of the respective server according to the given conditions.

$$\text{maxload } ES_k = \max CL[ES_k] - \Delta CL[ES_k] \quad (4.1.11)$$

$$\text{minload } ES_k = \min CL[ES_k] + \Delta CL[ES_k] \quad (4.1.12)$$

The above equations defined the updated load after the task migration. However, we need to ensure that the load among server is equal but maxloaded server before the load balancing cannot be the minloaded server during the load balancing.

$$\max CL[ES_k] \neq \min CL[ES_k] \quad (4.1.13)$$

$$\min CL[ES_k] \neq \max CL[ES_k] \quad (4.1.14)$$

The migration will be continuing till the last possible migration fulfils the stopping condition ($x \leq 1$) or no server is contributing to improving the resource utilization of the system.

Performance Matrix: Afterwards, we calculate the QoS parameters by utilizing Cumulative Load (CL) as discussed below:

- i. Makespan: It is the total completion time required to finish the most recent task. It is time taken by the last task to finish its execution. The goal in many distributed systems is to minimize the makespan, as this leads to more efficient utilization of resources and faster completion of the overall workload. The Makespan time is calculated as follows:

$$MS = \max\{CL[ES_k] | k \in K\} \quad (4.1.15)$$

- ii. Flow time: It is the overall time needed to finish all the tasks in the system. Using a minimal flow time can speed up the resource's response time. The can be computed as the total of all task completion times computed as follows:

$$FT = \sum CL[ES_k] | k \in K \quad (4.1.16)$$

- iii. Resource Utilization: It is a multitude of strategies are employed to optimize resource utilization in edge computing. It is determined as the fraction of time for which resources are actively working out of the total available time. The Average Resource Utilization RU_k can be calculated as follows: Load upon individual servers is calculated by
Load upon individual RU_j servers is calculated by eq (4.1.17)

$$RU_j = (CL[ES_j])/MS \quad (4.1.17)$$

Average Resource Utilization RU_k is calculated by eq(4.1.18)

$$RU_k = (CL[ES_k])MS \times k \quad (4.1.18)$$

- iv. Level of Load Balancing (LLB): It is a metric used for quantifying the degree of load imbalance between allocated resources for task execution. It gives a formal measure of the workload distributed across these resources and how effectively the workload is balanced across the system. LLB can be calculated by using the standard deviation as follows:

$$LLB = 1 - \frac{\sqrt{\frac{1}{k}(\sum_{j=1}^k (RU_k - RU_j)^2)}}{RU_k} \quad (4.1.19)$$

4.1.3 The Proposed Algorithm

This section discusses the process in which the tasks are submitted to the edge servers by users at application layers for their execution upon edge servers within

the minimum completion time. The algorithm is divided into three phases, i.e., Task rearrangement, Resource Allocation, and Load Balancing presented in Phases 1, 2, and 3, respectively.

The whole algorithm is divided into three different phases as mentioned earlier. The first phase is the task rearrangement phase. This strategy is specifically designed for tasks that arrive dynamically, to prepare and organize them for subsequent execution. The core focus of our approach is to arrange the tasks based on their average weight, selecting those that are closest to the average size and reorganizing them accordingly. The task, which is close to the average weight of the task batch, will be selected and placed into the task queue. Since all tasks in the queue are of approximately the same magnitude, a uniform task distribution is initially achieved, ensuring that neither excessively large nor small tasks are included. However, when we have multiple tasks having the same deviation (d), the task with the greater size will always be given the priority. The same will be chosen in the task queue.

In the second phase, tasks are allocated to servers from the rearranged queue based on the minimum completion time. By prioritizing the execution of tasks with the shortest completion time, the flow time across servers is reduced, which in turn enhances overall resource utilization.

In the load balancing phase, first we identify the maximum loaded and the minimum loaded server. Thereafter, we identify the difference between them to perform the task migration. Migration will be performed from the maximum overloaded server to the maximum underloaded server. We employ a task migration strategy with conditions ($x = diff/2, y = diff/4$). To ensure an even distribution of load across servers, we transfer a load of $diff/2$ from the overloaded server to the underloaded one. The task selected for migration is the one whose size is closest to $diff/2$. If no such task exists, we then proceed to consider tasks that meet the next condition, $diff/4$. During the load balancing process, the maxloaded server will never become minloaded and vice versa. This also reduces unnecessary migrations. The process will be repeated till the stopping condition arises, i.e., $x \leq 1$. This stopping condition is performing well in the considered level of server capacity and task size. Subsequently, the system's makespan is minimized by assigning tasks to the most appropriate resources. This allocation strategy results in a more balanced workload distribution across the servers in many cases. Additionally, it has been observed that the need for task migration is minimized, as the servers are effectively loaded. In some cases, it may affect

Algorithm 2 The Proposed Algorithm

```
1: procedure :
2:  $\{1 \leq i \leq N, 1 \leq k \leq M\}$ 
3: Input parameters  $\{t_i, ES_k, C[ES_k], Size[t_i], W\}$ 
4: Output parameters  $\{CL, MS, RU_k, [t_i|ES_j|]\}$ 
5: Phase 1: Task Rearrangement
6: Step 1: Calculate Average weight by eq (1)
7: Step 2: Calculate  $d = |t_i - W| \in i$ 
8: Step 3: Find  $min(d)[|t_i - W|]$ 
9: Step 4: If  $d(t_i) == d(t_i + 1)$ 
10:   {
11:     if  $Size(t_i) > Size(t_i + 1)$ 
12:       Select  $(t_i)$ 
13:     else
14:       Select  $(t_i + 1)$ 
15:   }
16: Step 5: Rearrange tasks such that  $d(t_i) < d(t_{i+1})$ 
17: Phase 2: Resource Allocation
18: Step 1: For  $t_i \in i$ 
19:   For  $ES_k \in k$ 
20:     • Calculate  $ET(t_i, ES_k)$  by eq (3)
21:     • Calculate  $CL$ 
22: Step 2: Find  $ES_j$  for  $t_i \in i : CL(t_i|ES_j) \leq CL(t_i|ES_k) \in k \neq j$ 
23: Step 3: Calculate  $CL(ES_k)$  from eq (4).
24: Phase 3: Load Balancing
25: Start  $\{1 \leq i \leq N, 1 \leq k \leq M\}$ 
26: Input parameters  $\{[t_i|ES_j|]\}$ 
27: Output parameters  $\{[t_i|ES_l|], MS, CL, RU_k, LLB\}$ 
28: Step 1: Calculate  $CL \in ES_k$ 
29: Step 2: find max, min
30:   •  $max = max CL[ES_k] \in k$ 
31:   •  $min = min CL[ES_k] \in k$ 
32: Step 3: Calculate  $diff = max - min$ 
33: Step 4: Calculate  $x = diff \div 2$  and  $y = diff \div 4$ 
34: Step 5: Find  $S = CL(t_j) : CL(t_j) \leq x$ 
35:   Find  $t_i : CL(t_i) = max S$ 
36: Step 6: During Migration check the eq (13) and (14)
37: Step 7: Migrate  $t_i \rightarrow min$ 
38: Step 8: Repeat steps 4 to 8 until  $x \leq 1$  or  $S = \phi$ 
39: Step 9: Calculate MS,  $RU_k$ , CL, LLB by eq (15), (18), (4), (19), respectively.
40: end procedure
```

the performance of the system, so to check the exact performance of the system, we tested the whole scenario multiple times with different generated input values. In all these scenarios, the server condition and the tasks characteristics are changing during the execution. In this way, it is expected that our algorithm will outperform in most of the cases as compared to existing heuristics.

4.1.4 Time Complexity Analysis

The time complexity of the proposed ABRL algorithm is primarily determined by the task rearrangement, resource allocation, and load balancing mechanisms. The task rearrangement algorithm sorts the tasks in terms of their sizes in comparison to the average task size and picks the closest values in an iterative process, leading to a time-complexity of $(O(N^2))$ where (N) is the number of tasks.

Similarly, the resource allocation step will allocate every task to the most appropriate edge server based on all the available (K) edge servers resulting in a time complexity of $(O(N \times K))$, where (K) is the number of edge servers. Similarly, the load balancing step entails determining maximum and minimum loaded servers, and task migration, also necessitating the assessment of tasks on servers, and adds a time cost of $(O(N \times K))$.

Constant time operations are made to update load values and to compute performance metrics. In this way, the most common computation in each task is the rearrangement and server evaluation, which leads to a total time complexity of $(O(N^2 + N \times K))$. The total time complexity of the proposed ABRL algorithm when taking into account the total (N) number of tasks is $(O(N^2 + N \times K))$. In large task scenarios $(N \gg K)$, it is the quadratic term that is important and can be approximated by $O(N^2)$ and in the most favorable situations, the time complexity becomes $(O(N \times K))$.

4.1.5 Illustrative Example

In this section, an example is given to show the demonstration of the proposed model. During the first phase, the task will arrive in a queue and store in a batch T_0 . Collection of all tasks making a batch of them and rearranging them in the queue will be formulated according to the proposed algorithm as discussed in Section 4.1.3. In this illustrative example, we consider 20 tasks $T_0 = t_0, t_1, t_2, t_3, \dots, t_{19}$ and 3 edge server $ES_k = ES_0, ES_1, ES_2$. The $C[ES_k]$ of all three servers are

4MIPS, 8MIPS, and 12MIPS for $k = 1, 2, 3$ respectively. The execution time of every task on each edge server is calculated by eq (3). The size of all tasks with execution time on each server is computed in Table 4.1.

Table 4.1: Task size and execution time on ES_k

Task-id	Task-size	ES_0	ES_1	ES_2
t0	49.23	12.3075	6.15375	4.1025
t1	15.17	3.7925	1.89625	1.264167
t2	18.54	4.635	2.3175	1.545
t3	52.12	13.03	6.515	4.3434
t4	49.93	12.4825	6.24125	4.6083
t5	86.16	21.45	10.77	7.18
t6	17.39	4.3475	2.17375	1.449167
t7	53.0	13.25	6.625	4.4167
t8	47.43	11.8575	5.92875	3.9525
t9	19.98	4.995	2.4975	1.665
t10	70.67	17.6675	8.83375	5.889167
t11	56.71	14.1775	7.08875	4.725833
t12	76.2	19.05	9.525	6.35
t13	46.22	11.555	5.775	3.851667
t14	66.06	16.515	8.2575	5.505
t15	95.05	23.765	11.88125	7.92083
t16	95.16	23.79	11.895	7.93
t17	48.46	12.115	6.0575	4.03833
t18	47.47	11.865	5.93375	3.955833
t19	67.15	16.785	8.39375	5.59583

In the Task Rearrangement phase, all tasks are collected and rearranged into the rearranged task queue by selecting the most nearby task size of the average size of all tasks. In this example, from all 20 tasks, the average is 53.90, so the nearest task size is 53 of t7. The next task is selected based on the task size that is nearest to the previous task size. The nearest size is 52.12 which is of t3, by following the same procedure and arranging them in a queue. The rearranged queue is $T = \{t_7, t_3, t_4, t_0, t_{17}, \dots, t_{15}, t_{16}\}$.

In the Resource Allocation phase, the tasks (t_i) are allocated to the best possible server (ES_k). Each task has its own execution time on each server as shown in Table 4.1. Now we take the first task from the rearranged task queue and give it to the server that has minimum completion time. For the first task, the ready time is considered zero. The first task is t_7 in the rearranged queue and executes on ES_2 because ES_2 has the highest computation power among all

servers and also consumes minimum execution time for t_7 , i.e., 4.4167. The next task in the rearranged task queue is t_3 , and ES_2 will take a minimum completion time. The completion time of t_3 on ES_2 is 8.8601 (by adding the execution time of a previously executed task). For ES_1 and ES_0 , the completion time and execution time are the same because the ready time is zero for both of them. The execution time for t_3 is 6.515 and 13.03 on ES_1 and ES_0 respectively. ES_1 is selected for the execution because it will take a minimum completion time. So by following the same procedure, all tasks are allocated to all edge servers with the consideration of minimum completion time as shown in Figure 4.2.

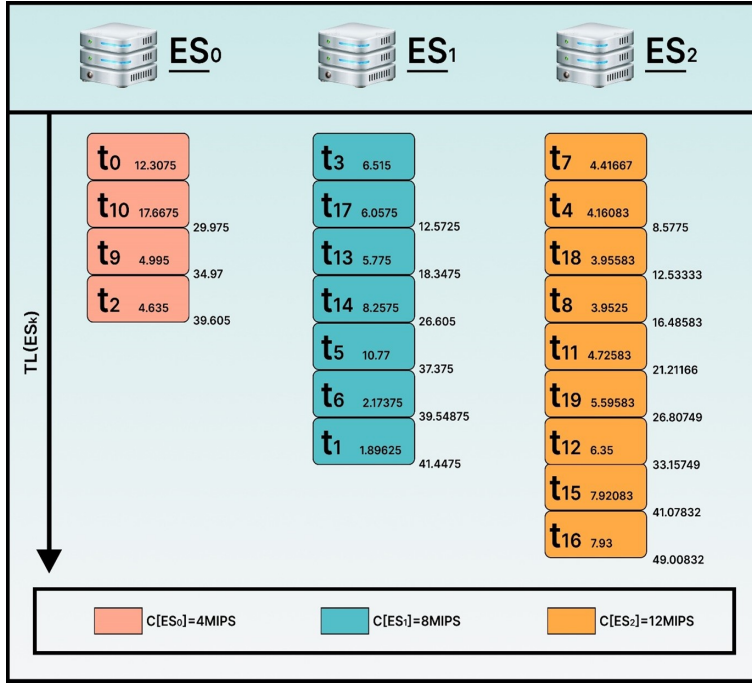


Figure 4.2: Flow Diagram of Resource Allocation to Edge Servers

After execution of all tasks, the completion time of ES_0 , ES_1 , and ES_2 is 39.605, 41.4475, and 49.0083, respectively. The last task will be executed on ES_2 ; therefore, the cumulative load of ES_2 is the makespan of the system, i.e., 49.00832. Average resource utilization (RU_k) is calculated by using the equation as shown below.

$$RU_k = (((39.605 + 41.4475 + 49.0083) \div (3 \times 49.0083)) \times 100) = 88.46\%$$

After the allocation phase, the obtained makespan and average utilization are 49.0083 and 88.46%. To achieve minimum makespan and enhance resource utilization, load balancing is needed across the servers. because ES_2 has the maximum load and ES_1 has the minimum load. To improve the efficiency load needs to

migrate from the maximum loaded ES_1 to the minimum loaded server ES_2 which is discussed in the next phase. In the load balancing phase, the migration of load

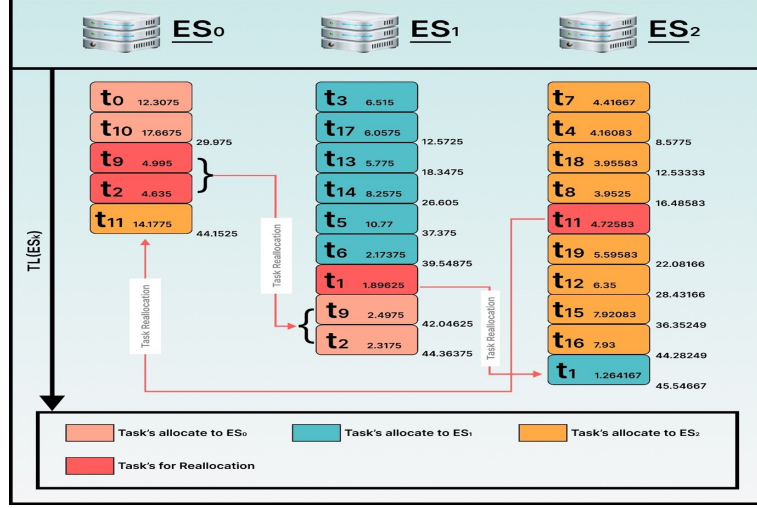


Figure 4.3: Flow of Load Balancing And Migrated Task's

from maximum loaded to minimum loaded server will be formulated. Firstly, we calculate the load difference between both servers. ES_2 has 49.0083 and ES_0 has 39.605. The difference in load is 9.403 between both servers. We migrate the tasks having execution time near to the $\text{diff}/2$. Therefore, we migrate the t_{11} from ES_2 to ES_0 . The same procedure we follow till the stopping condition rises as discussed in the proposed algorithm. The flow of task migration for load balancing is shown in Figure 4.3.

After the load balancing, the completion time of ES_0 , ES_1 , and ES_2 is 44.1525, 44.36625, and 45.54667, respectively. So the updated makespan of the system is 45.54667. And, the updated average resource utilization is calculated below:

$$RU_k = (((44.1525 + 44.36625 + 45.546677) \div (3 \times 45.546677)) \times 100) = 98.115\%$$

This illustrative example shows how the algorithm flow works to balance the load across the servers.

4.2 Experimental Study

In this section, we discuss the experimental setup, simulation environment, and the results we get from the execution of the proposed algorithm.

4.2.1 Experimental Setup

The results are tested on the "MacBook Air - Midnight" machine with an Apple M3 chip with 8-core CPU, 10-core GPU, 16-core Neural Engine, 16GB unified memory, and 256GB SSD storage. The MacOS operating system is used. The ODL controller is used as an SDN controller and integrated with the help of VMware.

The algorithm is implemented in Python 3.12. The simulation parameters are given in Table 4.2. The number of tasks varies from 100 to 10000 and is tested on both 20 and 50 servers. The task size and edge server power vary in the range given in the table.

Table 4.2: Configuration of task size and edge servers

Number of Tasks	100-10000
Number of Servers	20, 50
Task Size	100-3000 MI
Computing Capacity of Edge Server	10-100 MIPS
Parameters	Makespan, Resource Utilization, Level of Load Balancing

4.2.2 Results & Discussion

The proposed algorithm ABRL is compared with FCFS [66], SJF [67], DWECS [69], Max-Min, Min-Min [72], PTAL [70], and ACTE [71] algorithms on parameters makespan, average resource utilization, and level of load balancing. All considered algorithms are tested and implemented in the simulation environment for performance evaluation. To check the exact performance of algorithms, we executed them 10 times and then took their average results. Since Edge Computing is a distributed and heterogeneous environment. Therefore, at each instance, we may get the task with a random size. The task size and computing capacity of the edge server vary at each trial, as mentioned in Table 4.2. Therefore, if we take the result by considering only the one-time execution of the ABRL algorithm and other algorithms, that will not show the authenticity of the generated results. That is why we have taken the average of results generated 10 times randomly according to the range of different considered parameters.

The values in Tables 4.3 to 4.8 show the obtained makespan, average resource

utilization, and load balancing percentage level by existing techniques compared to the proposed algorithm. The positive values show the enhancement of results, which means the makespan decreases for a proposed algorithm and enhancement on the other two parameters, which is the main goal of the study. In some places, negative values persist, which shows an increase in the makespan for our algorithm, however, increases in the other two parameters. However, these instances are very few, with marginal differences only. The necessary observation and results discussion of existing techniques to the proposed technique on all three parameters are given below:

4.2.2.1 Observations

It is evident from Tables 4.3 to 4.8 and Figures 4.4 to 4.9 that the FCFS strategy obtains the highest makespan and lowest resource utilization and level of load balancing among all the compared techniques. SJF slightly performs better than FCFS and excels in the considered parametric values. On the other hand, the combination of FCFS and SJF with deadline, i.e., DWECS optimizes the parameters more than SJF and FCFS. ACTE strategy has slightly better results than Min-Min for all evaluated parameters. PTAL somewhat optimizes QoS parameters as compared to Min-Min and ACTE. It is also clear from the results, that the proposed ABRL algorithm surpasses the three existing algorithms, i.e., Min-Min, ACTE, and PTAL for all considered cases and conditions with a substantial difference. The Max-Min outperforms for a small number of tasks ranging from 100 to 2000 for 50 servers. However, for a large number of tasks, i.e., 5000-1000, ABRL surpasses all considered approaches for both 20 and 50 servers. For 20 servers, ABRL outperforms all considered heuristics for various cases and conditions.

Table 4.3: Percent deviation of makespan for proposed approach from optimal value for 20 servers

20 Servers							
No. of tasks	FCFS	SJF	DWECS	Min-Min	ACTE	PTAL	Max-Min
100	31.11756	27.96533	24.51076	19.29492	17.13231	14.90052	4.727324
250	7.719114	7.123122	6.519382	4.964501	3.972103	6.932604	7.089361
500	6.486308	5.628632	4.755079	3.082722	2.893332	4.399192	0.37902
750	14.2775	12.49655	10.64002	7.851419	7.420802	5.723296	1.296523
1000	6.181629	5.422213	4.650402	3.82219	2.602898	2.761174	2.020373

2000	1.579748	1.351752	1.122696	0.78042	0.711596	0.704994	0.05853
5000	0.636912	0.542875	0.44866	0.318817	0.293136	0.229593	0.0179
10000	0.205746	0.18577	0.165787	0.143104	0.058795	0.18212	0.144699

Table 4.4: Percent deviation of makespan for proposed approach from optimal value for 50 servers

50 Servers							
No. of tasks	FCFS	SJF	DWECS	Min- Min	ACTE	PTAL	Max- Min
100	16.4626	13.26082	9.803822	7.074977	2.390166	4.32027	-36.0751
250	30.87332	28.06463	25.01804	17.08688	14.06058	30.57438	-1.08281
500	7.363752	6.34137	5.296169	1.951067	1.191913	12.03966	-7.06375
750	5.582423	4.852763	4.111738	2.93351	2.630783	2.826727	-0.62016
1000	6.472117	5.608308	4.728395	2.726652	2.237984	6.485571	-1.92558
2000	3.147518	2.704542	2.257495	1.238004	1.188063	2.943228	-1.00139
5000	2.080375	1.787831	1.493534	1.072054	1.044098	0.693816	0.180157
10000	0.962542	0.826824	0.690734	0.47409	0.444941	0.461747	0.01783

Table 4.5: Percent deviation of average resource utilization for proposed approach from optimal value for 20 servers

20 Servers							
No. of tasks	FCFS	SJF	DWECS	Min- Min	ACTE	PTAL	Max- Min
100	74.36872	56.97876	42.74287	30.27631	21.27957	15.45923	0.247869
250	22.13455	18.2822	14.66545	9.857746	7.644443	9.509307	-0.80281
500	9.932078	8.327856	6.769782	4.675876	3.921466	3.774774	-0.3803
750	22.02507	18.12218	14.46121	11.25589	7.485773	5.722291	0.060701
1000	4.302273	3.546818	2.802228	2.136231	1.29754	0.532717	-0.7596
2000	2.298405	1.948904	1.601782	1.160283	0.961323	0.815866	-0.0525
5000	3.80187	2.840339	1.653926	0.464455	0.393543	0.276026	0.01451
10000	2.701411	2.267679	1.219003	0.167675	0.189494	0.2741	0.197948

Table 4.6: Percent deviation of average resource utilization for proposed approach from optimal value for 50 servers

50 Servers							
No. of tasks	FCFS	SJF	DWECS	Min- Min	ACTE	PTAL	Max- Min

100	38.59994	32.83787	30.16539	21.15273	9.030133	5.880402	-33.1053
250	60.3727	45.20446	32.65755	23.39958	12.31907	3.708978	-7.51177
500	20.87692	16.81591	13.0189	5.682701	3.616847	14.73938	-6.76648
750	6.630495	5.687337	4.760717	3.018183	2.564435	4.978884	-0.00465
1000	11.49184	9.446494	7.474842	4.97459	3.67228	3.329273	-1.8366
2000	7.482705	4.31127	3.439503	2.212038	1.900593	1.440621	-0.93419
5000	4.610174	3.159285	1.748091	1.087863	0.990557	1.564797	0.14622
10000	2.195393	0.985662	0.812136	0.501576	0.403084	0.728412	-0.1522

Table 4.7: Percent deviation of level of load balancing for proposed approach from optimal value for 20 servers

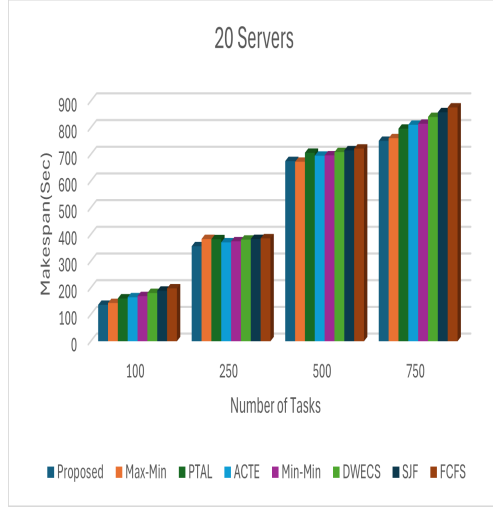
20 Servers							
No. of tasks	FCFS	SJF	DWECS	Min- Min	ACTE	PTAL	Max- Min
100	109.7761	79.39941	56.70733	50.12125	16.52941	13.85057	0.121298
250	19.63626	16.1017	12.76999	11.03529	5.113215	4.165573	-0.35141
500	9.481821	7.884907	6.333909	5.681289	2.413379	2.016401	-0.20549
750	11.44092	9.535193	7.693551	6.907613	3.358799	2.365107	0.280843
1000	4.794633	3.995132	3.207737	2.92184	1.141855	0.839829	-0.26399
2000	2.325837	2.011437	1.698963	1.391771	1.147324	0.902217	0.435734
5000	0.813669	0.687343	0.561333	0.495307	0.257513	0.175788	0.00096
10000	0.774893	0.412225	0.353401	0.298835	0.253133	0.202803	0.132486

Table 4.8: Percent deviation of level of load balancing for proposed approach from optimal value for 50 servers

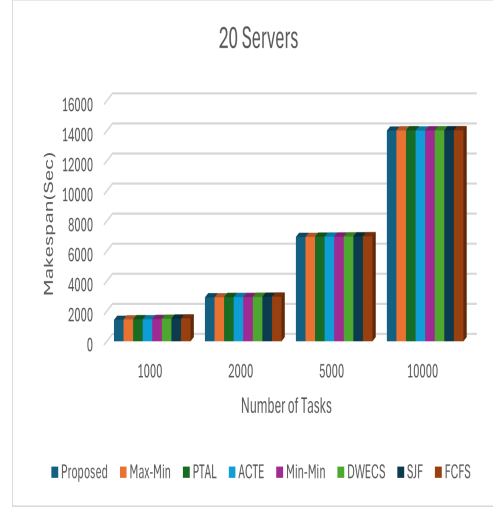
50 Servers							
No. of tasks	FCFS	SJF	DWECS	Min- Min	ACTE	PTAL	Max- Min
100	175.9758	149.5811	118.9765	90.51927	15.98562	25.96562	-34.62
250	90.89217	64.60687	44.68426	40.60474	5.31257	5.757066	-9.44089
500	19.28494	15.24725	11.47396	9.564913	1.762453	2.078499	-4.84307
750	15.65969	13.09659	10.64463	6.522669	8.141554	5.228689	0.090425
1000	11.11982	9.206627	7.358195	6.316142	2.704542	2.744997	-0.6215
2000	5.270004	4.420117	3.583842	3.062201	1.489827	1.443754	-0.16038
5000	2.059805	1.740579	1.423344	1.207547	0.637613	0.609585	0.02722
10000	1.100282	0.928507	0.757314	0.669467	0.303253	0.291996	0.02552

Makespan:

The comparative analysis of makespan on 20 and 50 servers are give below:

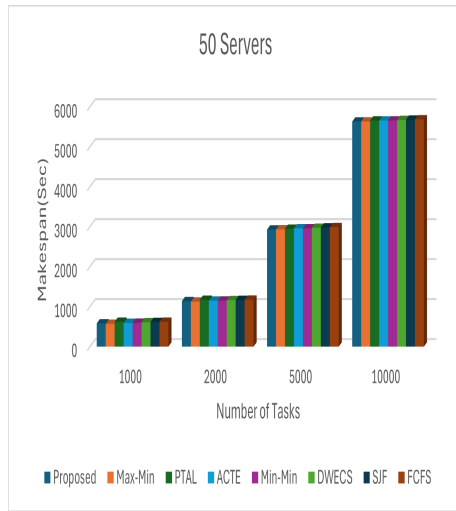


(a) 1

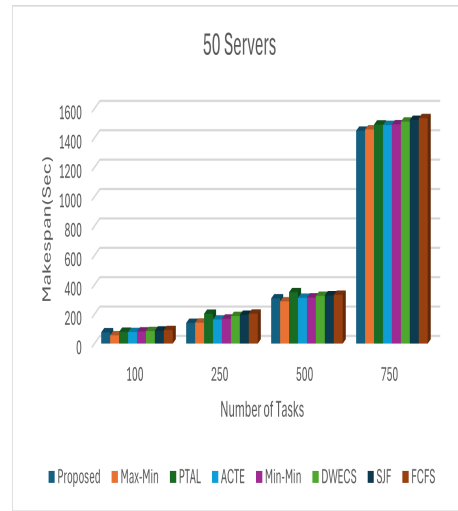


(b) 2

Figure 4.4: **1.** Makespan results on 20 servers for task sizes 100-750, **2.** Makespan results on 20 servers for task size 1000-10000



(a) 1



(b) 2

Figure 4.5: **1.** Makespan results on 50 servers for task sizes 100-750, **2.** Makespan results on 50 servers for task size 1000-10000

Average Resource Utilization:

The comparative analysis of Average Resource Utilization on 20 and 50 servers are give below:

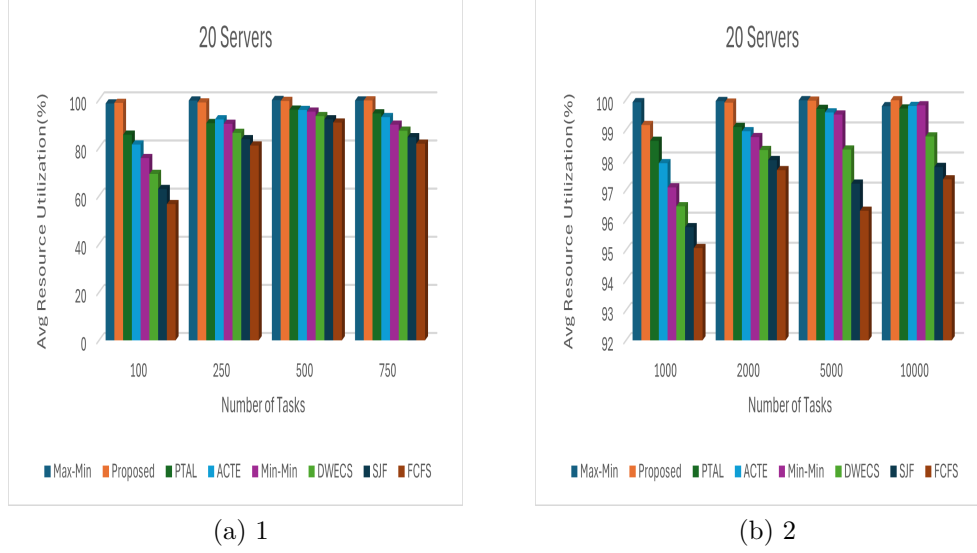


Figure 4.6: **1.** Average Resource Utilization results on 20 servers for task sizes 100-750, **2.** Average Resource Utilization results on 20 servers for task size 1000-10000

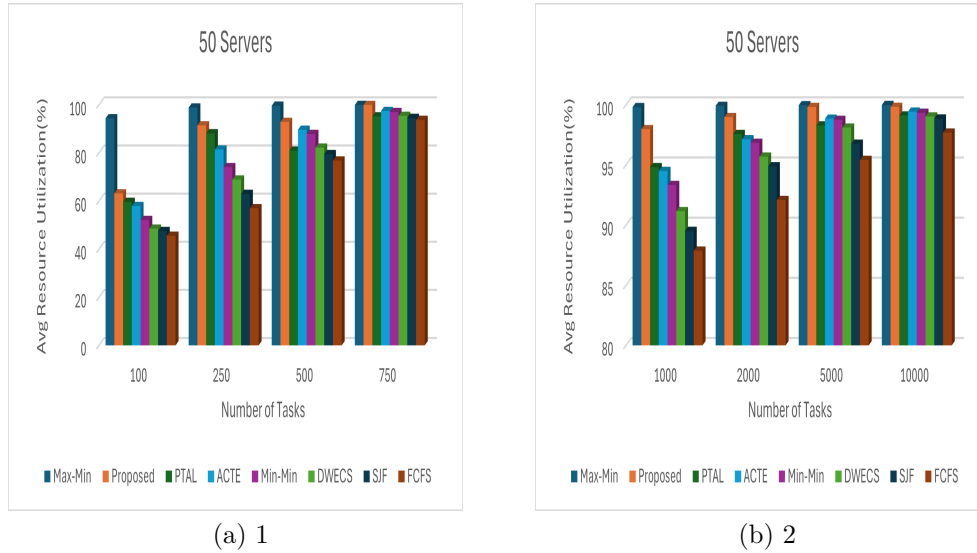
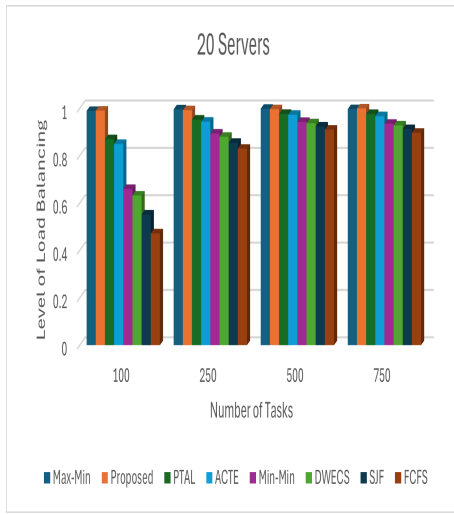
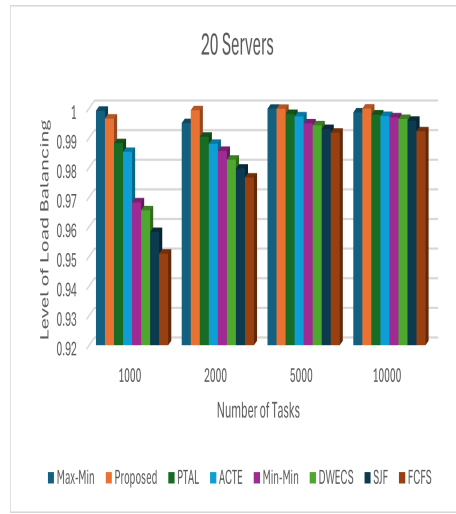


Figure 4.7: **1.** Average Resource Utilization results on 50 servers for task sizes 100-750, **2.** Average Resource Utilization results on 50 servers for task size 1000-10000

Level of load balancing:

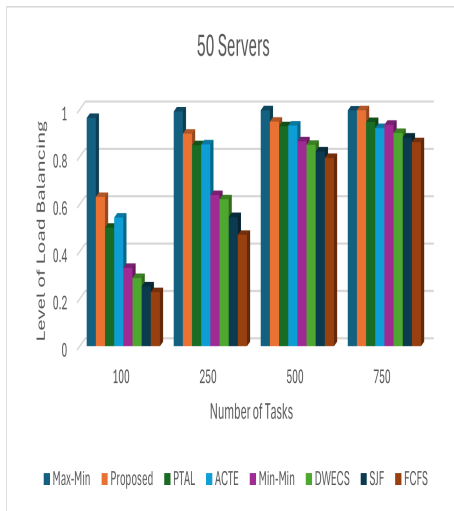


(a) 1

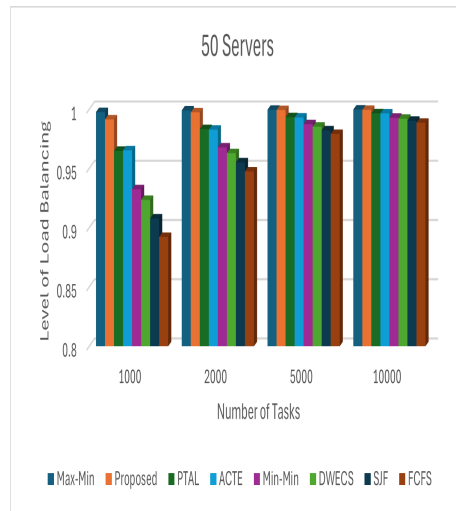


(b) 2

Figure 4.8: **1.** Level of load balancing results on 20 servers for task sizes 100-750, **2.** Level of load balancing results on 20 servers for task size 1000-10000



(a) 1



(b) 2

Figure 4.9: **1.** Level of load balancing results on 50 servers for task sizes 100-750, **2.** Level of load balancing results on 50 servers for task size 1000-10000

4.2.2.2 Performance Analysis

As shown in the results, FCFS performs poorly compared to all considered algorithms. In FCFS, the tasks come in an arbitrary order and are allocated to the resources, leading to an increase in the waiting time. The resources are allocated on a first-come, first-served basis. FCFS is a non-pre-emptive algorithm that might cause starvation problems. The SJF algorithm has similar issues. In SJF, the shortest task is executed first, leading to the starvation problems for the long processes. Further, it cannot deal efficiently with a mixed workload. Another recently proposed DWECS, i.e., the combination of SJF, EDF, and FCFS. Though this approach completes the task within the deadline, however, it suffers from maximization in TAT and burst time.

Min-Min surpasses FCFS, SJF, and EDF due to its strategic allocation technique. It arranges the task list in ascending order and schedules the smallest task first. The largest task is allocated to the resources at the end based on availability only. ACTE is more effective than the Min-Min. The load migration and distribution technique are inadequate in ACTE. Also, after a threshold the task migration is simply stopped, which could be handled in a better way to optimize the load distribution. In some instances, ACTE excels at PTAL as well. PTAL outperforms ACTE, Min-Min, and other compared heuristics. However, underperformed Max-Min and the proposed ABRL. PTAL uses a minimum completion time strategy for task scheduling. The load balancing strategy utilized by PTAL can be enhanced if we can improve the stopping condition of load balancing.

From the studies, we have found that Max-Min is the highest benchmark in resource utilization. It arranges the task queue in descending order according to the task size. Furthermore, it allocates the largest task to the server having the highest execution capacity so that at last the queue has a smaller task remaining. Therefore, it helps in improving the efficiency and performance of systems and provides better resource utilization.

In the proposed ABRL algorithm, the allocation of tasks is based on the average task size and allocates them to the largest capacity server having minimum completion time. It tries to cover more tasks in migration due to efficient stopping conditions, therefore enhancing system performance. During execution, we have found that in some cases the proposed algorithm performs nearly equal to the max-min and beats this highest benchmark. Even for some instances, our proposed algorithm performs better than max-min. However, there are some limitations observed in the proposed algorithm as well.

4.3 Contextual Summary

The integration of SDN and edge computing is a novel combination, simultaneous network monitoring and user response execution occur. While monitoring the network, any potential issues/disruptions can be identified that have to be addressed promptly to minimize downtime and maximize productivity. SDN management enhances the effective use of all edge resources. The proposed ABRL algorithm provides a foundation for optimizing resource utilization, reducing makespan, and ensuring efficient load distribution across edge nodes. It can offer insights into how computational tasks can be allocated based on various factors like server capabilities, network conditions, and task sizes. The algorithm is specially designed for the edge environment dealing with heterogeneous and decentralized environments. If we see it practically, then the algorithm can improve the performance and reliability of edge networks by ensuring that tasks are allocated optimally, thus enhancing overall system performance. In edge environments with limited resources and diverse device capabilities, it can help avoid resource bottlenecks and ensure that workloads are appropriately distributed, leading to better scalability and adaptability.

In the proposed algorithms the assumption of the communication cost being negligible is based on the proximity-driven feature of the edge computing paradigm and centralized SDN-based architecture used in the proposed system. Since, all the computation is carried out near the data source in a localized edge environment, there is a reduction in transmission distances and hence low latency and low communication time.

Besides, this paper utilizes the advantages of SDN and edge computing. A distributed paradigm is proposed to support proficient resource utilization. The ABRL scheme is proposed for optimized resource allocation and load balancing, working in different phases. It collects the user's tasks in a queue and further rearranges them in another queue by calculating the average weight of all tasks. The task having task sizes near the average weight size will be selected in the rearranged queue. Afterward, the tasks are assigned to the resources that can execute them at the earliest. This approach also works with balancing the load based on the load on the server.

The proposed scheme aims to minimize makespan and enhance resource utilization

and level of load balancing to ensure the edge computing system's performance. The proposed approach is compared with recent and existing state-of-the-art techniques for varying numbers of tasks and servers at different heterogeneity levels. The experimental results indicate that the proposed scheme outperforms almost all considered algorithms and is highly effective in achieving the desired optimized results for all considered cases.

Chapter 5

Dynamic Priority Task Scheduling for Resource Optimization in SDN-based Edge Computing

The emergence of edge computing has brought transformative potential to the way computation is performed at the network's edge. In contrast to the traditional cloud computing paradigm, edge computing is local to data sources, which enhances computational latency, reduces bandwidth usage, and minimizes computational overhead for time-sensitive applications. Nevertheless, the complexities of real-time environments and the increased heterogeneity of edge devices have become the sources of multiple challenges in efficiently managing the necessary computational resources.

In real-world edge environments, problems such as uneven workloads on edge servers, insufficient computing power, and fluctuating task arrival rates often lead to load imbalances, prolonged execution times, and reduced QoS. Moreover, the absence of global visibility and real-time decision-making in traditional distributed systems further complicates task scheduling and resource allocation.

To address these concerns, SDN has been integrated into edge computing environments due to its centralized control and global network view. By facilitating the monitoring and coordination of network resources, SDN enables the dynamic reconfiguration and efficient routing of these resources. Nevertheless, integrating SDN with edge-computing requirements presents complexities related to delays in communication among controllers and nodes, overhead from dynamic resource allocation, and challenges in achieving load balance due to high workload variance

and heterogeneity across different infrastructures.

In response to these challenges, this chapter proposes a three-layer SDN-based edge computing architecture and a novel DPTS algorithm. The proposed architecture integrates user devices, SDN-controlled edge servers, and categorized VMs to support task offloading and resource utilization. In the case of a task creation by an IoT device, the SDN controller will select the most appropriate edge server, depending on proximity and current occupancy. The chosen server will then transfer the job to a VM (one with the most appropriate characteristics and processing requirements for executing the job).

In the proposed DPTS algorithm, a dynamic priority management process is suggested. Individual tasks are then assigned three coefficients, which are added to determine the final priority score. On that basis, tasks are then prioritized and scheduled for prompt execution, ensuring that crucial tasks are completed efficiently and maintaining overall system responsiveness. The algorithm can dynamically adjust to different task loads and heterogeneous VM capabilities, thus remaining efficient even in the most dynamic scheduling environments. The following are the contributions.

- Introduced the DPTS algorithm, which is specifically designed for handling tasks in edge infrastructure environments.
- Subsequently, we assign VM to the tasks effectively in order to maximise efficiency through a dynamic priority approach that adjusts as tasks arrive.
- A three-layered SDN-based model is also proposed that integrates IoT devices, edge nodes, and virtual machines to fulfil the requirements of the distributed systems.
- To show the scalability of the proposed approach, we considered low and high heterogeneity among tasks and high heterogeneity among VMs. It also ensures that the model adapts well to varying computational demands, enhancing SDN dynamism.
- Later, we devised a server selection mechanism to identify the most suitable server based on task origin, aiming to optimize latency.
- A VM categorization method was introduced to classify VMs according to specific task requirements, distributing workloads efficiently and ensuring load balance across computational resources.

- The detailed experimental evaluation of the proposed DPTS algorithm was conducted on an SDN-based model for various cases and conditions.
- Key performance metrics, including Makespan, Total Completion Time, Average Resource Utilization, latency, and Level of Load balancing were analysed and compared within an edge computing environment.
- Finally, the results are compared with the recent similar approaches, i.e., the Optimal IoT Service Offloading (OSO) [73] scheme, Utility-Aware Task Offloading (UTO) [75], and First Fit (FF) [74].

The OSO method performs very well in modest scaling, but its performance degrades as the job scale increases. The integration of NSGA-2 for multi-objective optimization incurs computational overhead. Still, there is a need to enhance scalability, adaptability, offloading mechanisms, and processes to reduce overhead. Similarly, UTO involves multiple access points that collect data and send it to the server, but there is no proper mechanism to handle the massive volume of data from these access points. As a result, there is an increase in underutilized and overutilized machines, affecting the system's overall resource utilization. Besides, the FF algorithm might work in certain limited or small-scale systems. However, it is suboptimal for large, heterogeneous, and dynamic environments like cloud data centers. The lack of scalability, energy conservation, support for server heterogeneity, load balancing, and failure management in FF makes it unsuitable for such complex systems. Additionally, current models generally overlook critical factors such as task and resource heterogeneity, model dynamism, and balanced load distribution, all of which are essential for effective edge computing.

5.1 System Model

In modern edge infrastructure and SDN environments, efficiently managing and allocating computational tasks across distributed resources is critical to improving system performance, resource utilization, and overall scalability. However, achieving optimal task scheduling in such systems continuously remains a challenging problem due to the dynamic nature of cloud workloads and the need for real-time decision-making. To overcome the issues, the system model is discussed, which comprises system architecture, problem formulation, and proposed algorithm. The proposed system is designed to support the task scheduling strategy and focus on the enhancement of resource optimization.

5.1.1 System Architecture

The proposed algorithm is a resource allocation strategy designed for SDN-based edge computing. To support the algorithm, a system architecture is designed that is divided into three layers discussed below:

Application Layer: This layer is the topmost layer of system architecture. This layer contains several IoT devices deployed at different locations and generates tasks for the user that need execution. It acts as an interface that connects the user to the system directly so that the user can send the request and get a response from the system in the form of executed tasks.

Control Layer: It is the middleware of the system that contains the SDN controller. The SDN helps to manage the network and connect the data-creating devices deployed at the application layer and edge servers present at the host layer. The functions and components of the control layer are discussed below.

- i Network Monitoring: This component is responsible for monitoring the network. It stores the information of actively connected IoT devices to the network. It also manages the bandwidth of the system, which enhances the system performance. With managing the bandwidth, it also considers the factors responsible for task propagation from one point to another. This information on network and IoT devices helps to optimize resource allocation and improve the performance of the system.
- ii Priority calculator: It helps to detect the priority level of tasks generated from IoT devices and adds a priority coefficient according to the proposed algorithm. This determines the priority of tasks and helps to make better scheduling decisions accordingly.
- iii Task Sorter: After calculating the priority of the tasks, the task sorter will sort the tasks in descending order. The tasks will be sorted according to their priority and size.
- iv Cluster Formulator: According to the information received from the task sorter, it helps to formulate the clustering of specific task ranges at various edge nodes. The task size and other related information are used to formulate the cluster, deploy the tasks on different edge nodes.

- v Task Queue Generator: It prepares the queue according to their task size, based on calculated priority and requirements from the application layer for smooth dispatch of tasks to edge nodes present at the host layer.
- vi Queue Manager: It manages the queues based on the information on the control layer components. It ensures the tasks are dispatched to the edge nodes in an optimized sequence and maintains proper flow from the application layer down to the edge layer.
- vii Resource Allocator: It allocates the resources to computational tasks across edge nodes to meet the demand from the application layer. It provides the resources for tasks according to the proposed algorithm that meets QoS requirements for each application.

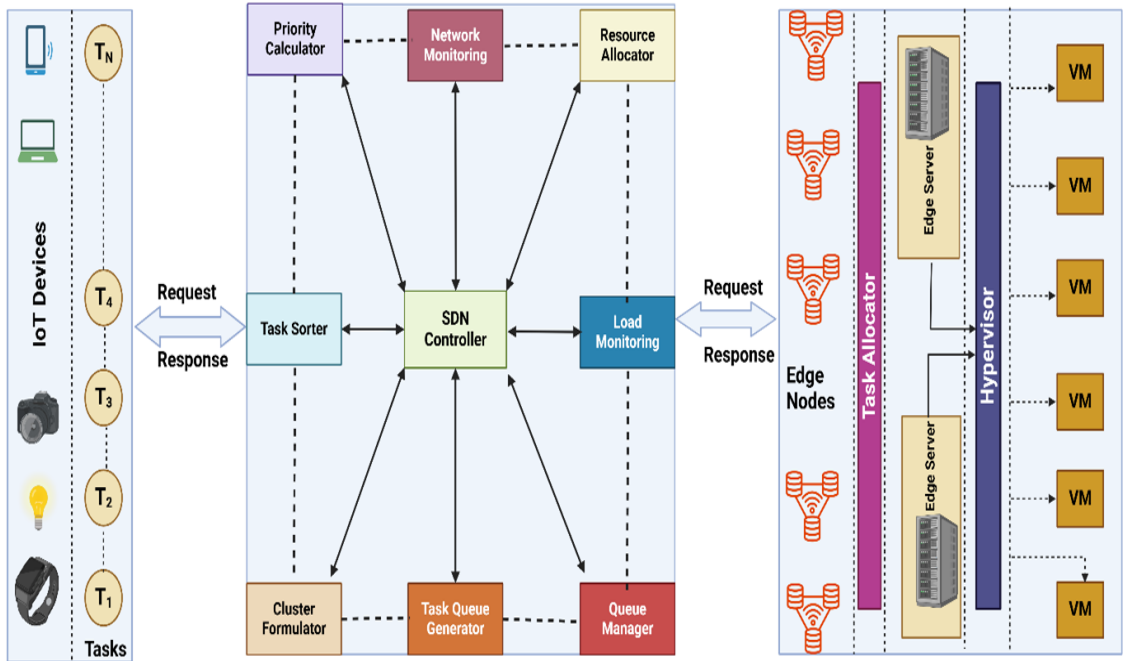


Figure 5.1: Proposed Architecture for SDN-based Edge Computing

Host Layer: The third layer of the proposed infrastructure contains edge nodes and virtual machines built upon the physical edge servers. This layer is responsible for the smooth execution of tasks generated from the application layer closer to the source of data. The edge nodes are responsible for data capturing/storage through the control layer. The edge nodes store tasks of different sizes coming from the task queue of the control layer. Further, it allocates virtual machines to the tasks according to the proposed algorithm.

5.1.2 Problem Formulation

The current SDN architecture under consideration consists of servers, each equipped with several edge nodes and VMs. In this setup, each node serves as a buffer where tasks are queued before being dispatched to VMs for execution. The allocation of tasks from nodes to VMs must be performed in a manner that ensures optimal workload distribution, minimizes makespan, total completion time, and maximizes resource utilization and level of load balancing.

A Central controller, i.e., SDN is implemented to manage the interaction between servers, nodes, and VMs. The controller's responsibility is to:

- **Monitoring:** Monitor the tasks loads on the servers and nodes.
- **Resource allocation:** Dynamically allocate the tasks from nodes to VMs based on current system conditions. It ensures that the system achieves efficient resource allocation while minimizing total completion time, preventing bottlenecks, and avoiding overloading any single component. However, several challenges arise that are discussed below.
- **Load Balancing:** How can tasks be effectively distributed across VMs to avoid server or VM overloading while maintaining system efficiency?
- **Task Scheduling:** What is the optimal way to schedule tasks from nodes to VMs in real-time, considering varying workloads and the need for minimal waiting time.
- **Resource Utilization:** How can resource utilization be maximized without causing performance degradation or task delays?
- **Transmission Time:** How can the architecture transmit the task flow?
- **Task heterogeneity:** Dynamism in task heterogeneity includes the significant challenge in the architecture.

Therefore, the problem to be addressed in this problem is to develop an effective and uniform SDN-based task scheduling mechanism that optimizes resource utilization, minimizes waiting time for tasks, and ensures smooth management of computational resources under dynamic workloads. The proposed system architecture and controller need to address these challenges, ensuring seamless task flow and real-time adaptation to varying system conditions.

5.1.3 Problem-Defination

Assume that there are (S) edge servers deployed in the network to process the incoming tasks. In this system, each task (t_i) is initially allocated to its corresponding edge node N deployed with an edge server. Moreover, the additional characteristics of the considered system are presented below:

- The system has S number of edge servers.
- Each server has several nodes specifying and storing the task ranges based on the task size.
- Each node is associated with a dedicated VM which is responsible for executing the tasks assigned to that specific node.
- The incoming task is allocated to the nearest server to reduce the transmission time.
- The system considers three priority coefficients, i.e., α, β, γ , and assigns them to every task randomly.
- The final allocation is performed based on both priority and task size of the task.
- The system considers low and high task heterogeneity benchmarks.
- Bandwidth is 500 MB per Second.

5.1.3.1 Objective

The primary goal is to design an efficient task scheduling algorithm in an SDN-based edge computing architecture, where tasks are stored at multiple edge nodes and processed by VM on the edge servers. The algorithm aims to minimize the Makespan and Latency and maximize Resource Utilization while considering task priority and task size into account.

5.1.3.2 System Overview

The Edge computing environment consists of:

I. **Edge Servers:** A set of S edge servers that are geographically distributed to handle incoming tasks/requests from multiple IoT devices. Each server contains:

- **Nodes:** Each server has N nodes represented as:

$$N : \{n_1, n_2, \dots, n_n\} \quad (5.1.1)$$

Where incoming tasks are initially stored.

- **Virtual Machines (VMs):** Each node is associated with several VMs (V), equal to the number of nodes in the system, and represented as:

$$V : \{v_1, v_2, \dots, v_n\} \quad (5.1.2)$$

Where every virtual machine has the capacity defined in MIPS (Million Instructions Per Second)

- **Placement:** The geographical location of the edge server.

II. **Tasks:** There is a set of T tasks represented as:

$$T : \{t_1, t_2, \dots, t_n\} \quad (5.1.3)$$

Where each task has the following specifications:

- **Size $S((t_i))$:** A value that denotes the computational requirement of the task t_i defined in MI (Million Instructions).
- **Initial Priority (IP):** A randomly assigned priority coefficient (α, β, γ) based on the urgency/importance of the tasks for execution.
- **Final Priority (FP):** The final priority is the calculated priority based on the Initial Priority and the size of the task.

$$[FP(t_i) = S(t_i) + P(t_i) \quad \forall t_i \in T] \quad (5.1.4)$$

- **Origin (O):** This is the source of the task, which is used to route it to the nearest edge server. It could be any device.

The priority coefficients in the DTPS are empirically determined to minimise computational load and ensure real-time flexibility. Systematic or multi-objective

tuning is less feasible in latency-sensitive edge environments due to dynamic, unpredictable task arrivals. The chosen weights offer a balanced trade-off between performance and scalability, as observed in experiments. The proposed model addresses the management and execution of tasks across multiple VMS within an SDN-based architecture in five distinct phases. The phases of the model are formulated as follows:

Phase 1: Server Selection

When a task originates, the model selects the nearest server to the task's origin to minimize latency and other transmission overheads. By dynamically choosing the closest server, this phase reduces the time required for data transfer and enhances response times, improving the system's overall performance.

Phase 2: Task Range Selection/Node Selection

Given a dynamic set of incoming tasks, each with unique computational requirements, the model must determine the K subset of tasks to be processed within a specific time window. The goal of this phase is to balance the load effectively by ensuring the feasible range of tasks is selected based on the incoming task size. The model selects the node for the incoming task by dividing the highest expected task size by the number of considered nodes, as shown below:

$$[K = \frac{Highest_{TaskSize}}{n}] \quad (5.1.5)$$

Where, $Highest_{TaskSize}$ is $\begin{cases} 100 & \text{for low task heterogeneity} \\ 3000 & \text{for high task heterogeneity} \end{cases}$ And n is the number of nodes considered in the system.

Phase 3: Task Storage

Selected tasks are then stored within designated edge nodes. These nodes serve as temporary storage, maintaining tasks until they are ready for VM allocation. This phase requires ensuring task integrity and managing storage efficiently to prevent bottlenecks, considering reliability metrics.

Phase 4: VM Allocation

Within each node, tasks are arranged in the order of their final priority value, with higher values scheduled first. The tasks coming from the tasks queue after sorting are ready to assign to the VM now. The objective of this phase is to minimize makespan by effectively matching tasks to suitable VMs based on factors such as VM capacity, current load, etc.

Phase 5: Task Execution

Once tasks are assigned to VMs, they proceed to execution. This phase requires monitoring and managing task progress. This step is crucial for meeting the required QoS while optimizing resource utilization.

Every VM has its busy time/ready time indicating the history of tasks executed by the VM. The Busy time of all VMs is initially zero. After every VM assignment, the Execution time of the task, $E(t_i)$ is calculated as:

$$[E(t_i) = \frac{S(t_i)}{C(v_j)}] \quad (5.1.6)$$

Once the task is completed over the assigned VM, the busy time of the corresponding VM is updated as:

$$[BT(v_j) = BT(v_j) + E(t_i)] \quad (5.1.7)$$

5.1.3.3 Formulating the Optimization Variables

In an edge computing environment, efficient task scheduling requires a detailed estimation of task completion time across various components. To achieve this, the total completion time for each task can be calculated by combining the transmission time, propagation time, and execution time. This formulation provides an accurate assessment of end-to-end task completion, makes decisions about task assignment to minimize latency and maximize resource efficiency, and supports the specifications of the edge environment.

- I. *Transmission Time*: This represents the time required to transmit the task data from the task origin to the edge server.

$$TransmissionTime = \frac{S(t_i)}{Bandwidth} \quad (5.1.8)$$

- II. *Propagation Time*: This is the time it takes for the data to propagate over the distance between the task origin and the server.

$$PropogationTime = \frac{Distance\ between\ O(t_i)\ and\ corresponding\ Edge\ Server}{Speed} \quad (5.1.9)$$

- III. *Task Size ($S(t_i)$)*: The data size of a given task in the million instructions.
- IV. *Transmission Rate (TR)*: The rate at which data is transmitted between the task origin and the server in Mb/s.
- V. *Distance to Server (D)*: The physical distance between the task origin and the assigned edge server.

- VI. *Propagation Speed* (PS): The speed at which the signal propagates through the network.
- VII. *Bandwidth*: The amount of data uploaded per unit of time.
- VIII. *Execution Time* (ET): The time taken to execute the task on the server once it arrives, measured in seconds.

5.1.3.4 Objective Functions

- I. *Minimize Makespan* (M): The Makespan is the total time required to execute all tasks. The objective is to minimize this value:

$$\text{Makespan } (M) = \text{Max} [BT(v_j)] \quad \forall \quad t_i \in T. \quad (5.1.10)$$

- II. *Maximize Resource Utilization*: Involves efficient management of SDN edge resources like storage, bandwidth, and optimal task handling.

$$\text{Resource Utilization } (U) = \frac{\sum_{j=1}^n BT(v_j)}{\text{Makespan} \times \text{Number of VMs}} \quad (5.1.11)$$

- III. *Minimize Latency*: The mean total time taken for a task to be transmitted, propagated, and waiting within the edge computing network. It is calculated by summing the transmission time, propagation time, and waiting time for each task.

$$\begin{aligned} \text{Latency} = & \text{Transmission Time } (t_i) + \text{PropogationTime } (t_i) \\ & + \text{WaitingTime}(WT_i) \end{aligned} \quad (5.1.12)$$

- IV. *Minimize Total Completion Time* (TCT): The cumulative time required for data transmission, propagation, waiting and execution.

$$\begin{aligned} TCT(t_i) = & \text{TransmissionTime } (t_i) + \text{PropogationTime } (t_i) \\ & + \text{WaitingTime}WT_i + E(t_i) \end{aligned} \quad (5.1.13)$$

- V. *Maximize Level of Load Balancing* (LLB): LLB is the normalized value and ranges between 0 and 1 where 0 indicates poor load balancing and 1 indicates perfection in load balancing. It assesses how evenly tasks are distributed across the computed resources and is computed as follows:

$$LLB = 1 - \frac{\sigma}{U} \quad (5.1.14)$$

Where σ is the Standard Deviation of resource utilization across VMs and U is Average Resource Utilization.

5.1.4 The Proposed Algorithm

The algorithm aims to efficiently manage and execute tasks across multiple nodes and edge servers by prioritizing and optimizing task assignments based on task characteristics and available resources. The steps outlined below detail the process from task division to performance evaluation.

I. *Task Division:*

Objective: Divide a total of T tasks into K ranges according to the number of available nodes.

Method: This helps to distribute the workload evenly across the nodes, ensuring that no single node is overloaded, and resources are utilized effectively.

II. *Task Origin Detection:*

Objective: Identify the origin $O(t_i)$ of each task t_i .

Method: Each task is assigned to the nearest edge server, minimizing latency and improving response times by utilizing the most accessible resources.

III. *Task Assignment to Nodes:*

Objective: Assign each task to a specific node within the selected edge server based on its size $S(t_i)$.

Method: This ensures that larger tasks are directed to nodes that can handle them effectively, balancing the load across the system.

IV. *Assign Priority:*

Objective: Randomly assign priority coefficients to each task from a pre-defined set of values α, β, γ .

Method: This introduces variability in task execution priorities, allowing the system to adapt to different situations and optimize performance.

V. *Final Priority Calculation:*

Objective: Calculate the final priority $FP(t_i)$ for each task.

Explanation: The final priority is the sum of the task size and its assigned priority coefficient, allowing for more significant tasks to be prioritized.

VI. *Task Sorting:*

Objective: Sort the tasks within each node in descending order of their final priority.

Method: This sorting ensures that the most important tasks are executed first, improving overall efficiency and resource utilization.

VII. ***VM Simulation:***

Objective: Simulate n virtual machines per server and sort them based on their processing speed in ascending order.

Method: By understanding the capabilities of each VM, tasks can be assigned to the most suitable VMs, optimizing execution times.

VIII. ***Task Execution:***

Objective: Execute tasks in each node according to their sorted order.

Method: Tasks are assigned to VMs based on their speed, ensuring that faster VMs handle higher-priority tasks, thereby reducing execution time.

IX. ***Performance Evaluation:***

Objective: After task execution, evaluate overall system performance.

Metrics: Calculate makespan, resource utilization, total execution time, and level of load balancing.

Importance: This evaluation helps assess the efficiency of the algorithm and identify areas for potential improvements.

5.1.5 Time Complexity Analysis of DPTS

The **DPTS** consists of multiple phases, each contributing to the overall computational complexity. Below is a detailed complexity analysis:

- **Task Division:** Each task from T is assigned to a node, leading to a time complexity of $O(T)$.
- **Task Origin Detection:** The origin of each task is identified and assigned to the nearest edge server. This requires $O(T)$ operations.
- **Task Assignment to Nodes:** In this phase each task is allocated to a node within the selected server based on task size resulting in $O(T)$ complexity.
- **Priority Assignment:** Priority coefficients are assigned and incur $O(T)$ time.
- **Final Priority Calculation:** Computing the final priority for each task contributes an additional $O(T)$.

Algorithm 3 Dynamic Priority Task Scheduling Algorithm (DPTS)

```
1: Input:  $T$ : Total tasks,  $n$ : Total nodes,  $S[t_i]$ : Task size of  $t_i$ ,  $P[i]$ : Priority
   coefficient of task  $t_i$ ,
2:       VMs: Set of VMs on each server (sorted by ascending capacity),
   Bandwidth
3: Output: Makespan, Resource Utilization, Total Time
4: // Task Division
5: for each task  $t_i \in T$  do
6:   Assign  $t_i$  to a node  $N$  based on a predefined range of tasks per node
7: end for
8: // Task Origin Detection
9: for each task  $t_i \in T$  do
10:  Determine origin  $O[t_i]$ 
11:  Assign  $t_i$  to the nearest edge server based on  $O[t_i]$ 
12: end for
13: // Task Assignment to Nodes
14: for each task  $t_i \in T$  do
15:  Assign  $t_i$  to a node based on task size  $S[t_i]$  to balance node workload
16: end for
17: // Priority Assignment
18: for each task  $t_i \in T$  do
19:  Randomly assign priority coefficient  $P[t_i] \in \{\alpha, \beta, \gamma\}$ 
20: end for
21: // Final Priority Calculation
22: for each task  $t_i \in T$  do
23:  Compute final priority  $FP[t_i] = S[t_i] + P[t_i]$ 
24: end for
25: // Task Sorting by Final Priority
26: for each node  $N$  do
27:  Sort tasks in  $N$  in descending order of  $FP[t_i]$ 
28: end for
29: // VM Simulation and Sorting
30: for each server do
31:  Simulate  $J$  VMs with varying processing capacity ( $J = n$ )
32:  Sort VMs in ascending order of capacity
33: end for
34: // Task Execution
35: for each node  $N$  do
36:  for each task  $t_i$  in  $N$  (sorted by  $FP[t_i]$ ) do
37:    Assign  $t_i$  to the next available VM
38:    Execute  $t_i$  on the assigned VM
39:  end for
40: end for
41: // Performance Evaluation
42: Calculate Makespan: Time required for the last task to complete
43: Calculate Resource Utilization: Average VM usage
44: Calculate Total Time including transmission and propagation
```

- **Task Sorting by Final Priority:** The tasks in each node are sorted based on final priority, resulting in a complexity of $O(T \log T)$, (Quick Sort)
- **VM Simulation and Sorting:** Here, each server is simulated into J VMs, which requires the time complexity $O(J)$. Simultaneously, the VMs are sorted, which requires the Complexity $O(J \log J)$. Thus, the total complexity of this step will be $O(J \log J)$.
- **Task Assignment:** T tasks are allocated to the next available VM leading to the complexity of $O(T)$.
- **Overall Complexity:** The dominant terms are $O(T \log T)$ and $O(J \log J)$ from task sorting and VM sorting respectively. Since $T \ll J$ in large-scale systems, the overall complexity is $O(T \log T)$.

5.2 Experimental Study

This section implements experiments within our proposed system framework to evaluate our method. This section discusses an experimental simulation setup for assessing results, conducting a comparative analysis, and comparing the results with existing algorithms

5.2.1 Experimental Setup

The experimental setup was conducted using Mininet in Ubuntu, with an ODL controller integrated through Python. This configuration was deployed on an Apple MacBook Pro M3 Max with an M3 chip featuring a 14-core CPU, a 30-core GPU, and a 16-core neural engine. The system further includes 36GB of unified memory and 1 TB of SSD storage, operating under the MacOS environment.

Table 5.1: Simulation Setup and Parameter

Description	Value
Number of IoT devices	100, 200, 300, 400, 500
Task Size	For Low Heterogeneity: 1- 100 Million Instructions (MI) For High Heterogeneity: 100-3000 MI
Number of VMs	7 Per Edge Server
VMs Computation Power	10-100 Million Instructions Per Second (MIPS)

Bandwidth/ Transmission Rate	500 MIPS
Parameters	Makespan, Resource Utilization, Level of Load Balancing, Total Completion Time
channel Power	$2 * 10^8 m/s$

In this system, the IoT devices transmit requests/tasks for execution. The task will be stored at the edge node, however, the purpose of positioning edge nodes close to the edge server is solely to collect data and transmit the tasks in the suggested way to the virtual machines within the edge server. By following the proposed algorithm, the edge nodes directly connected to the main edge server offload the tasks to it. The considered simulation parameters are discussed in the given Table 5.1. Additionally, the simulation sample depicting the deployment of server and origin of upcoming tasks over the network is presented in Figure 5.2.

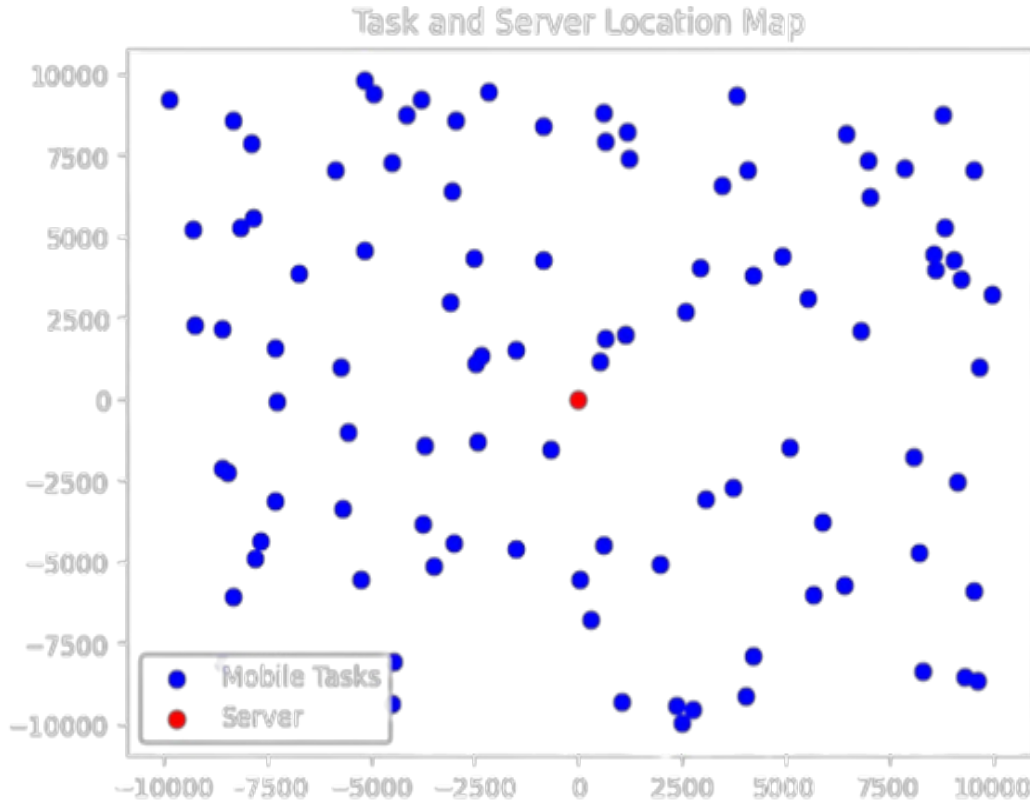


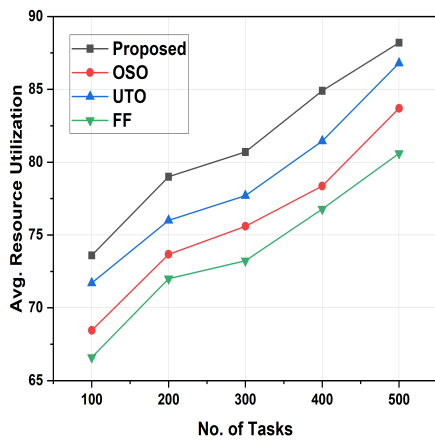
Figure 5.2: Simulation Sample of tasks and server deployment

5.2.2 Results and Discussions

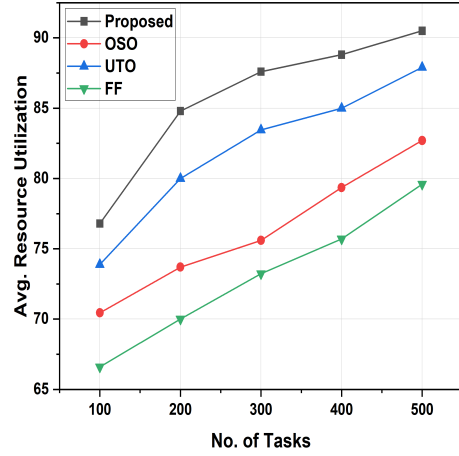
The proposed algorithm is compared with OSO, UTO, and FF based on the makespan, average resource utilization, total completion time, and level of load balancing. The algorithm is tested and implemented in the specified environment for performance evaluation. We test each experiment 10 times, then average the results on different generated task sizes and VM computation power using the values presented in the table. The number of tasks can range from 100 to 500, and their sizes can vary from 1 to 100 MI for low heterogeneity and 100-3000 for high heterogeneity. The computational power of VMs can vary from 10 to 100 MIPS, usually considered high heterogeneity. We assume that the edge server deploys the edge nodes. The time from IoT devices to edge nodes is calculated using the channel power. The time between the server and the nodes is considered negligible in this case. We deploy the entire infrastructure within an area of 10 km to 10 km, placing servers and devices in randomly generated positions. In the process of resource allocation, the proposed algorithm aims to minimize the total execution time and makespan and maximize the average resource utilization and level of load balancing.

Average Resource Utilization

It is the standard parameter to measure how effectively resources are utilized and it is computed across edge nodes and VMs. It is calculated by the busy time of active VMs on the edge server directly connected with the edge nodes. It tells the system's efficiency and performance with various numbers of tasks and task sizes. Figure 5.3 depict the average resource utilization on low heterogeneity and high heterogeneity levels, respectively. It is evident from Figure 5.3.1 that at low heterogeneity, the proposed algorithm increases the resource utilization with 3.26% from UTO, 7% from OSO, and 10% from FF. Figure 5.3.2 analyzes that at high heterogeneity, DPTS performs better at 4.4% from UTO, 11.65% from OSO and 16.53% from FF. The analysis shows that for the cases of extreme load, the proposed algorithm optimizes resources in all instances. Besides, FF performs worst in all scales of tasks at both heterogeneity levels. The analysis shows that for the cases of extreme load, the proposed algorithm optimizes resources in all instances. Besides, FF performs worst in all scales of tasks at both heterogeneity levels. The limitation with the FF is that the tasks are not being scheduled efficiently, which can increase the inefficient use of resources.



(a) 1



(b) 2

Figure 5.3: **1.** Average Resource Utilization for Varying Number of Tasks(Low Heterogeneity), **2.** Average Resource Utilization for Varying Number of Tasks(High Heterogeneity)

The UTO performs better than OSO and FF because of its allocating strategy that effectively uses the resources at the edge. The OSO has a fixed scheduling and allocation system due to the adaptation of NSGA-2. Moreover, it distributes the load arbitrarily among resources and mainly focuses on minimum completion time. The proposed algorithm considers the efficient server selection mechanism and effectively distributes the load considering the nature and requirement of the system. Therefore, enhancement of average resource utilization is achieved.

Makespan

Makespan calculates the completion time of the last executed tasks on the VM. It specifies the last virtual machine that is running for task completion. When the makespan increases, it means a particular VM is working for a long time while others are idle, which negatively affects resource optimization. The proposed algorithm performs better in all considered task ranges at both heterogeneity levels. The proposed DPTS reduces the makespan by 11.67% to OSO, 23.54% to UTO, and 31.85% to FF with low task heterogeneity. In case of high task heterogeneity, the proposed method performs 9.6% better than OSO, 15.54% better than UTO, and 20.4% better than FF. The FF performs the worst on every scale of tasks and VM heterogeneity. When the task size is gradually increasing, it might affect the performance of the system because the tasks have a larger execution time. Even if VM is in process for a long time, the proposed algorithm optimizes the makespan

as compared to other algorithms.

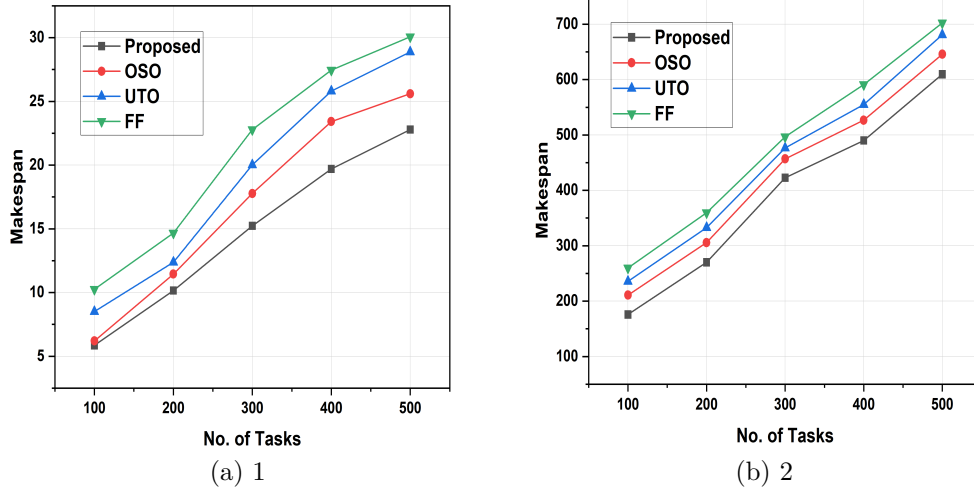


Figure 5.4: **1.** Makespan for Varying Number of Tasks(Low Heterogeneity), **2.** Makespan for Varying Number of Tasks(High Heterogeneity)

The proposed DPTS is specially designed for the optimization of the network and resources. The scheduling and allocation process is unified to optimize for makespan. On the other hand, FF doesn't focus on the uniform distribution of load. UTO somehow considers the virtualization and scheduling pattern across all servers, further performing better than FF. However, UTO also has the problem of uneven load distribution when single machines have multiple tasks for execution and require higher completion times. The OSO dynamically routes the tasks to the least loaded edge nodes. It performs better than the UTO and FF. The comparative graphs on makespan with both heterogeneities are shown in Figure 5.4.

Latency

It is the delay or time it takes for tasks to travel from the origin point to the server. It consists of transmission time, propagation time and waiting time. The lower the latency, the lower the response time, and that helps to enhance the performance of the systems. Edge computing is a complementary technology of cloud computing that came into light to have lower latency. If the latency is higher in the edge environment, it affects all parameters, performance and QoS. The proposed DPTS reduce the latency by 8.75% compared to OSO, 13.39% compared to UTO and 21.21% compared to FF.

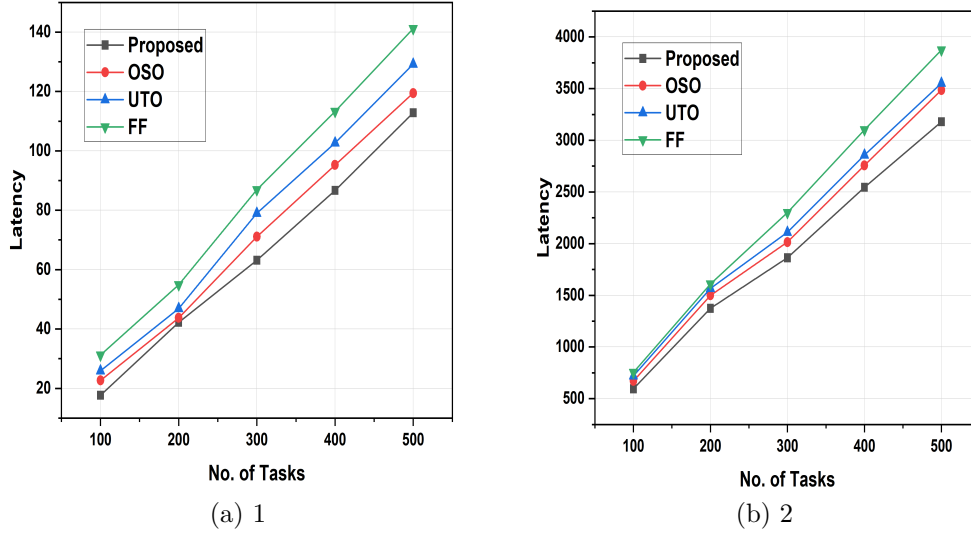


Figure 5.5: **1.** Latency for Varying Number of Tasks(Low Heterogeneity), **2.** Latency for Varying Number of Tasks(High Heterogeneity)

In case of latency and total completion time, there is only a difference in execution time. The OSO performs better than the UTO and FF because of its server selection mechanism that helps to minimize the time entity. The transmission time and propagation time are almost the same for everyone, but the allocation strategies differ with the waiting time, which affects the latency of the system. The UTO performs better than the FF but worse than the OSO.

UTO focus on the scheduling strategy that helps to enhance the average resource utilization but not able to categorize the tasks among several VMs so that may increase the waiting time. FF performs worse among all algorithms because in the strategy of this algorithm, several tasks wait for a long time due to its higher latency.

The proposed DPTS algorithm categorizes the task based on size and priority and provides a high-power virtual machine to the tasks that have a large task size, so it will help to provide resources to the tasks as early as possible. A comparative analysis of all algorithms on the the latency parameter is shown in the figure 5.5.

Total Completion Time

The total completion time consists of transmission time, propagation time, waiting time and execution time. This shows how effectively a job is completed within the stipulated time. The high total execution time is decreasing the performance and user experience of the distributed systems. Figures 5.6 shows that the proposed

algorithm minimized the total completion time of tasks on every scale and in both heterogeneity cases.

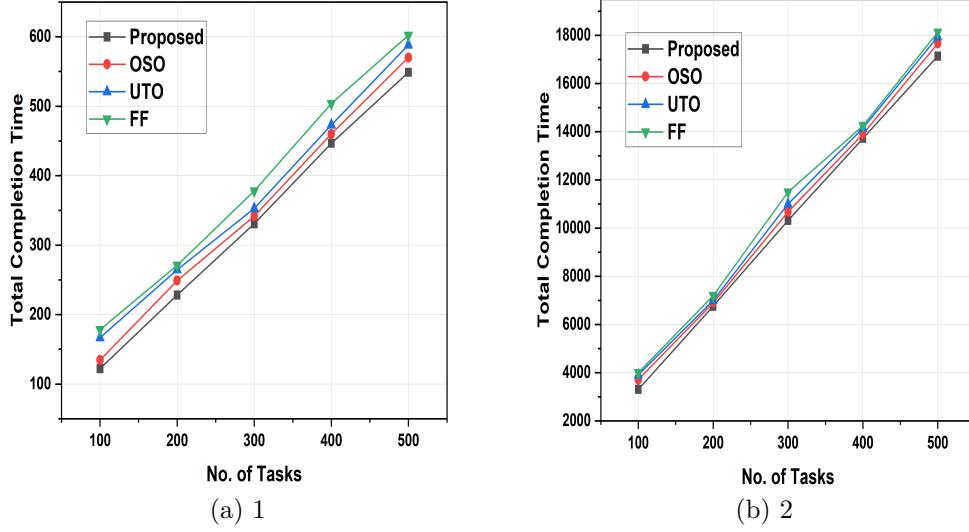


Figure 5.6: **1.** Total Completion Time for Varying Number of Tasks(Low Heterogeneity), **2.** Total Completion Time for Varying Number of Tasks(High Heterogeneity)

In low heterogeneity, the algorithm optimizes completion time by 5.48% than OSO, 11.8% than UTO, and 16.01% than FF. In the case of high heterogeneity, tasks are large and take a high transmission time and execution time. The proposed algorithm reduces the completion time by 4.09% to OSO, 6.498% to UTO, and 8.58% to FF. The OSO follows dynamic scheduling of tasks and focuses on the distribution of load when the resources are more utilized. OSO performs better than UTO and FF. The UTO focuses on prioritizing efficient resource usage, however, tasks require more processing time for execution.

The proposed DPTS algorithm maintains the trade-offs between resource usage and task completion time. It utilizes the resources efficiently and fulfills the dynamic resource requirements of the incoming tasks. It categorizes the tasks according to their priority and distributes them to VMs adequately so that task completion time will be minimized as compared to the other algorithms. The comparative analysis is shown in Figure 5.6.

Level of Load Balancing

The Level of Load Balancing describes how efficiently task distribution is among the computational resources considered edge nodes and VMs. In the case of non-

uniform distribution, there might be some overloaded or underloaded VMs that affect the performance of the system. The higher level of load balancing leads to a more satisfactory load distribution.

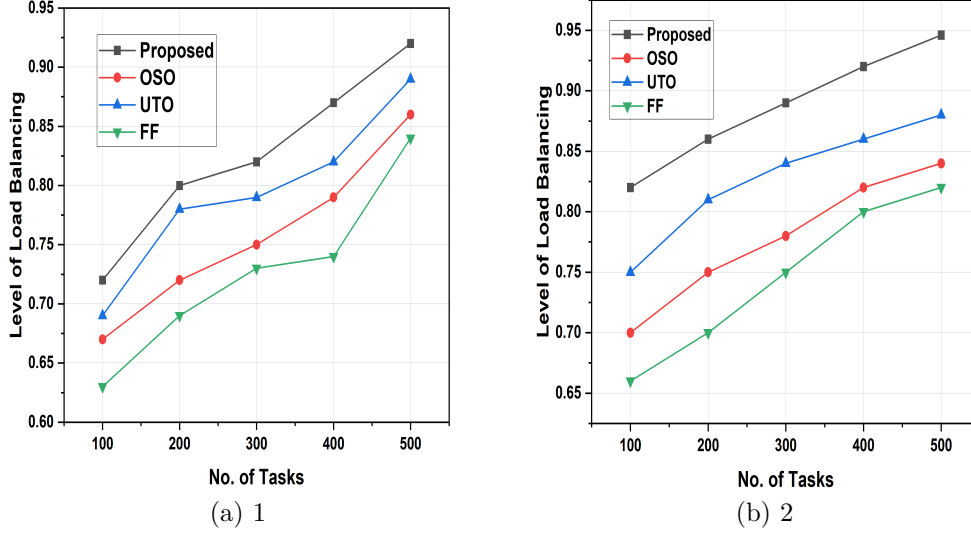


Figure 5.7: **1.** Level of Load Balancing for Varying Number of Tasks(Low Heterogeneity), **2.** Level of Load Balancing for Varying Number of Tasks(High Heterogeneity)

The proposed algorithm offers a superior load balancing level. The algorithm performs better than 4.03% to OSO, 9.02% to UTO, and 13.92% to FF in low heterogeneity cases. No doubt when the task size is small, the algorithm can distribute the load efficiently. However, it increases when dealing with high task heterogeneity. In this case, the proposed algorithm optimizes the level of load balancing 7.2% better than OSO, 13.54% better than UTO, and 18.80% better than FF. The FF is not able to perform an equal distribution of load due to an arbitrary selection of resources according to the first available fit. When the load balancing is effective, there is high resource utilization and low total completion time in UTO. The proposed algorithm performs better, especially on high heterogeneity of tasks. The comparative analysis is shown in Figure 5.7.

5.3 Contextual Summary

The integration of SDN and Edge Computing plays a crucial role in enhancing resource management by enabling flexibility and utilizing distributed and centralized systems over the network and computational resources. In this paper, we

investigated the task scheduling problems within an SDN-based edge computing environment and focused on proposing a solution to optimize resource utilization and minimize task completion time. We proposed a DPTS algorithm to address these issues, managing task prioritization and enhancing task scheduling efficiency. To successfully implement DPTS, we designed a three-layer SDN-based edge architecture that integrates IoT devices, edge nodes, and virtual machines to support efficient resource distribution across the network.

Additionally, the SDN controller has a global perspective of the network and supports efficient routing and management of resources further reducing communication overhead between edge servers. The main goal of the research is to maximize the metrics of computing performance, including the makespan, resource use, and load balancing, the model does not emphasize the communication aspects. Also, the postulation of high-speed connections and managed network conditions in the edge infrastructure will justify considering that communication delays have an insignificant effect on the overall system performance.

Experimental results demonstrate the model's effectiveness in optimizing makespan and total completion time, average resource utilization, and level of load balancing. However, its efficiency in real-time environments on extremely high task and VM numbers remains to be tested. Furthermore, the supplementary computations involved in assigning the dynamic priority may introduce some overhead, which could affect real-time environments.

Chapter 6

Conclusion and Future Advancements

In the evolving landscape of distributed computing, the integration of SDN with edge computing offers a strong solution to the challenges faced by traditional computing systems, resource management, and fair task scheduling. Edge computing helps reduce latency, improving responsiveness for time-sensitive applications because the computational resources are much closer to the data sources. However, since the edge environment is quite distributed and limited in resources, effective and responsive scheduling solutions must be developed to support system stability.

SDN, with its centralized control system and global network view, functions as an intelligent controller capable of dynamically managing and allocating resources among edge nodes. Moreover, SDN-based edge computing creates a hybrid infrastructure that combines localized computation with centralized orchestration, aiming to optimize resource use and enhance overall QoS. On the other hand, the significance of fair scheduling is a crucial factor in distributing computing resources equitably among diverse tasks and users. By leveraging SDN programmability and network-wide visibility, fair scheduling algorithms can adaptively allocate resources based on network conditions and application needs. It further addresses issues such as task starvation, resource monopolization, and bottleneck performance, thereby improving system efficiency and user satisfaction.

In this thesis, we initially reviewed some existing methods related to resource allocation, load balancing, and task offloading, and evaluated their advantages and disadvantages in the considered computing environment. We then studied the implementation of fair scheduling and resource optimization, including resource allocation, load balancing, and computation offloading, in SDN-based edge

computing systems. The focus of this work is to balance the load among edge servers. All servers allocate resources to the most suitable server, which enhances average resource utilisation and ensures the smooth execution of computational tasks. The introduced framework includes a system that helps prevent server overloads by redistributing tasks as needed, making the system more robust and quicker to respond.

To validate the effectiveness and scalability of the proposed model, extensive simulations were conducted using a custom-built Python platform integrated with the ODL SDN controller. Furthermore, a series of experimental studies were carried out under varying conditions using real-world-inspired task loads and diverse edge server configurations to assess the adaptability and reliability of the models. The evaluation outcomes consistently demonstrate improved performance in terms of makespan, resource utilization, total completion time, latency, and the level of load balancing factor, confirming the model's applicability in edge computing scenarios. Moreover, this thesis effectively addresses issues such as task starvation and uneven resource use, ensuring consistent QoS and compliance with SLAs.

6.1 Conclusion

The thesis begins with chapter 1 provides a thorough discussion of the evolving landscape of distributed computing, emphasizing the shift from centralized cloud environments to decentralized edge computing landscapes. As the demand for low-latency and high-throughput services increases, traditional resource management methods can no longer meet the requirements of real-time applications. This gap is addressed by edge computing, where computation resources are deployed near the data sources. However, this approach introduces new challenges related to resource heterogeneity, limited capacity, and network dynamics. To address these challenges, this chapter highlights the strategic integration of SDN with edge computing. The integration provides a global view of any network, making resource management more dynamic. The programmability of SDN in this edge computing model creates a hybrid platform that enhances flexibility, scalability, and service delivery.

Besides, fair scheduling is a crucial technique for ensuring the equitable allocation of resources on computing systems when there are competing tasks and users. However, fair scheduling is a traditional method that often suffers from

resource monopolisation and task starvation, leading to degraded QoS and violations of SLAs. Therefore, integrating fair scheduling is necessary to enhance the performance of SDN-based Edge Computing. By utilizing the centralized intelligence of SDN, fair scheduling can dynamically allocate resources based on the real-time network situation and the tasks' needs, which can improve the balanced use of the system.

Chapter 2 offers a thorough overview of recent advancements and current research on resource optimization in Edge Computing. It also discusses how SDN should be integrated into overall decision-making in these systems. SDN has been introduced as a valuable addition to edge computing, providing centralized management and programmability, which aligns with the dynamic and distributed nature of edge infrastructures. The literature is thoroughly reviewed and divided into four main areas of resource management in edge computing: load balancing, task scheduling, resource allocation, and computation offloading. Moreover, this chapter discusses the role of SDN specifically in enhancing these areas by offering a global perspective, dynamic control, and programmability. It also emphasizes the importance of architecture-level integration between SDN and emerging paradigms like NFV.

Based on critical analysis, it was identified that the literature requires the need to develop an effective scheduling and resource optimization model for an SDN-enabled edge environment. Further, the following ideas can be concluded:

- SDN has significant potential to address resource management challenges in edge environments because it offers flexibility, centralized control, and adaptability.
- The concepts of load balancing and task scheduling have been given significant attention as core components. But in multi-dimensional optimization, most of the information about latency, energy, and fairness remains unexplored.
- Further interest exists in using SDN alongside complementary technologies, such as NFV, to enable multi-provider orchestration and end-to-end service continuity.
- Most work functions in static or semi-dynamic environments; a more effective scheduling and orchestration approach is necessary in real-time, dynamic edge settings.

- Edge computing systems implemented using SDNs have shown positive results, such as reduced latency, better resource utilization, and scale-up.

Chapter 3 aims to improve resource allocation strategy in the SDN-based edge computing system by incorporating a hybrid optimisation scheme. The proposed Hybrid PSO-GWO model is introduced to enhance decision-making in resource allocation, reduce latency, and effectively use the energy resources. The hybrid approach will leverage the global search capabilities of PSO and the local convergence features of GWO to achieve high-performance scheduling. Tasks are distributed based on their computational needs, and resource selection is aimed at minimizing both time delay and energy consumption. This synergy enables parallel execution and optimized task assignment, thereby improving system responsiveness and reducing processing overhead.

The hybrid PSO-GWO model was evaluated using key QoS metrics such as time delay, energy consumption, and resource utilization. The results of the experimental analysis showed that the suggested model significantly outperformed both the individual PSO and GWO methods. The smallest time delay was measured at 0.00568, and the energy consumption was reduced to 0.00729, demonstrating the model's effectiveness in time-sensitive and energy-efficient edge-based settings. In addition, the results have been compared with baseline algorithms, consistently showing superior performance, with the hybrid PSO-GWO outperforming others across different operating loads and task levels. Therefore, the model has great potential as a solution for dynamic task scheduling, resource-efficient load balancing, and eco-friendly computation in the SDN-based edge environment.

In Chapter 4, the Average-Based Resource Allocation and Load Balancing (ABRL) algorithm is introduced for efficient task scheduling, resource allocation, and load balancing in SDN-based edge computing environments. The model utilizes the global perspective and control of SDN to dynamically schedule tasks and achieve optimal utilization of heterogeneous edge servers. ABRL operates through a structured multi-phase strategy. User-generated tasks are first loaded into a task queue. Then a rearrangement mechanism adjusts their order based on how closely their size matches the average weight calculated for all tasks. This redistributed queue facilitates an even distribution of the load across edge servers, depending on each server's readiness and demand levels. As a result, the time required to offload tasks is minimized, the overall makespan is reduced, and resource utilization is maximized.

The proposed model explicitly addresses load imbalance and underutilization is-

sues common in edge environments where devices are limited, distributed, and diverse. ABRL overcomes task starvation and server overloading, ensuring fair task division through its adaptive scheduling mechanism. The ABRL algorithm is implemented in Python, and its performance is evaluated by comparing it with various scheduling algorithms, namely FCFS, SJF, DWECS, Min-Min, Max-Min, ACTE, and PTAL. The evaluation considers key performance metrics, including makespan, average resource utilization, and load balancing efficiency. The experiments are conducted with task sizes ranging from 100 to 10,000, executed over two different server configurations comprising 20 and 50 heterogeneous edge servers. The experimental setup aims to assess the scalability and adaptability of the ABRL algorithm under diverse workload intensities and resource heterogeneity conditions.

Chapter 5 addresses the task scheduling issues associated with an SDN-enabled edge computing network. It proposes a new scheduling strategy that can balance resource availability and reduce task completion times. To achieve these goals, a DPTS algorithm is introduced, focusing on dynamic task prioritisation and improved scheduling efficiency. A three-layer SDN-based edge architecture is proposed to facilitate the use of DPTS, which includes IoT devices, edge nodes and virtual machines. This hierarchical structure simplifies task offloading and resource distribution in the edge network in a low-latency, efficient, and fast manner. The DPTS primarily focuses on reducing waiting time, as this is responsible for minimising latency, which is a primary parameter of edge computing.

The quality of the DPTS approach has been rigorously tested by comparing it with OSO, UTO, and FF on various parameters, including makespan, latency, total completion time, average resource utilisation, and load balancing degree, across different heterogeneity cases. The experimental results confirm the model's effectiveness in terms of workload distribution and resource allocation under various conditions. The experimental outcomes demonstrate that the proposed model achieves superior results in terms of workload distribution and efficient utilization of edge resources, ensuring improved system performance and responsiveness in diverse operational settings.

Chapter 6 acts as the main point of integration, bringing together the key outcomes and insights from the research presented throughout the thesis. It provides a comprehensive conclusion by summarising the main contributions, highlighting the effectiveness of the proposed models and algorithms, and emphasising their impact on task scheduling, resource allocation, and load balancing in SDN-

based edge computing environments.

6.1.1 Limitations

There are also possible limitations of proposed algorithms ABRL and DPTS in dynamic SDN-based edge computing systems. The limitations of algorithms are given below:

- The centralized SDN controller is an important part of the proposed approach which manages the tasks, allocates resources and makes load balancing decisions. This can provide visibility to a global network, but can create a single point of failure and scalability bottleneck in large-scale or highly dynamic edge environments.
- The use of heuristic-based decision mechanisms to select servers and prioritise tasks might not be optimal on a global scale, and thus lead to the sub-optimal allocation of resources and higher makespan in the presence of stochastic task arrival times.
- The algorithm is intended to deal with dynamic tasks, but the analysis is mostly performed with the simulated and controlled conditions. The edge environments in the real world are usually extremely unpredictable, can have network delays and the device mobility, and this can be seen to impact the performance consistency of the proposed model.
- The algorithms might fail to consider current network conditions, including bandwidth variations, packet jettison, and traffic jam that might influence the precision of transmission and propagation delay estimation.
- Load balancing stage entails the migration of tasks among edge servers to balance the workloads. The model however fails to put into consideration the migration costs in terms of delay in data transfer, network bandwidth consumption, and potential service interruption, which can affect the overall system performance in actual deployments.

6.2 Future Work

This chapter details several key directions for future improvements in SDN-enabled edge computing. The proposed enhancements concentrate on managing system

dynamism, real-time task scheduling, mobility-aware computation, and efficient task migration across diverse edge servers. These research avenues align with essential components of the proposed system, such as dynamic task scheduling, intelligent load balancing, and fault tolerance, within an edge architecture that integrates SDN controllers and IoT-driven environments. Future research can explore the following advanced scenarios:

- The integration of multiple distributed controllers is employed to mitigate faults within a centralised system.
- The proposed algorithm ABRL can be improved by considering the priority constraints and arrival times to ensure the proposed models operate effectively in large-scale, real-world edge environments.
- To address scalability and heterogeneity challenges in larger edge networks, future improvements to the proposed ABRL could involve dynamic, context-aware, and AI-driven methods that respond to real-time network conditions and resource availability.
- Future research can extend to include fault-tolerance methods like task replication, checkpointing and redundancy to guarantee fault-tolerant execution of tasks in the event of edge server or network failure.
- The suggested framework can be further expanded to incorporate real-time failure detection and recovery plans such as dynamic task migration and re-scheduling, to keep the system up and running in the face of node or link failure.
- Adding failure-aware decision-making to ABRL and DTPS can allow proactive redistribution of loads and enhance system resilience in edge environments characterized by extreme dynamism and susceptibility to failures.
- Security-aware scheduling, which takes into account the adversarial cases, such as malicious nodes and denial-of-service attacks, as well as the means of secure task offloading and trust-based node selection can be introduced into the future research.
- Future work could focus on managing highly dynamic task arrivals and departures in real-time edge scenarios. This includes accommodating bursty workloads, making decisions in real-time under strict deadlines, and addressing user behaviour.

- Consideration of more types of tasks, such as data-intensive and memory-intensive tasks, in SDN-based edge computing.
- Consider communication costs, overheads and load migration costs in the upcoming resource optimization strategies.
- The proposed model can be expanded by adding the dynamic voltage scaling mechanisms and both active and idle power consumption to the energy model in the future. It will also aim to perform a sensitivity analysis of PSO and GWO hyperparameters, including swarm size, inertia weight, and acceleration coefficients, to gain a better understanding of how these parameters affect the rate at which they converge to a solution and the quality of the solution.

Bibliography

- [1] Sadeeq, M.M., Abdulkareem, N.M., Zeebaree, S.R., Ahmed, D.M., Sami, A.S. and Zebari, R.R., IoT and Cloud computing issues, challenges and opportunities: A review. Qubahan Academic Journal, 1(2), pp.1-7.2021.
- [2] Hepsiba, C.L., Priyadarsini, R.J. and Suresh, S.A., . Integration of IoT and Cloud Computing Challenges: Survey. Integration, 8(01).2019
- [3] Qureshi, K., . Secure integration of IoT and cloud computing. marketing, 2(3), pp.4.2021
- [4] Stergiou, C., Psannis, K.E., Kim, B.G. and Gupta, B., 2018. Secure integration of IoT and cloud computing. Future generation computer systems, 78, pp.964-975.
- [5] Elazhary, Hanan. "Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions." Journal of Network and Computer Applications 128 (2019), pp.105-140.
- [6] Li, C., Zhang, Y., and Luo, Y. (2021). Deep reinforcement learning-based resource allocation and seamless handover in multi-access edge computing based on SDN. Knowledge and Information Systems, 63, pp. 2479-2511.
- [7] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, Mobile edge computing potential in making cities smarter, IEEE Communications Magazine, vol. 55, no. 3, pp. 38-43, 2017.
- [8] A. Filali, A. Abouaomar, S. Cherkaoui, A. Kobbane and M. Guizani, "Multi-Access Edge Computing: A Survey," in IEEE Access, vol. 8, pp. 197017-197046, 2020, doi: 10.1109/ACCESS.2020.3034136.

- [9] N. Kumar, S. Zeadally, and J. J. P. C. Rodrigues, "Vehicular delay-tolerant networks for smart grid data management using mobile edge computing," *IEEE Commun. Mag.*, vol. 54, no. 10, pp. 660, Oct. 2016.
- [10] S. Dama, V. Sathya, K. Kuchi, and T. V. Pasca, "A feasible cellular Internet of Things: Enabling edge computing and the IoT in dense futuristic cellular networks," *IEEE Consum. Electron. Mag.*, vol. 6, no. 1, pp. 66-72, Jan. 2017.
- [11] Yu, Wei, et al. "A survey on the edge computing for the Internet of Things." *IEEE access* 6 (2017): 6900-6919.
- [12] W. Yu et al., "A Survey on the Edge Computing for the Internet of Things," in *IEEE Access*, vol. 6, pp. 6900-6919, 2018, doi: 10.1109/ACCESS.2017.2778504.
- [13] W. Yu, G. Xu, Z. Chen, and P. Moulema, "A cloud computing based architecture for cyber security situation awareness," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2013, pp. 488-492.
- [14] Z. Chen et al., "A cloud computing based network monitoring and threat detection system for critical infrastructures," *Big Data Res.*, vol. 3, pp. 10-23, Apr. 2016.
- [15] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2016, pp. 1451-1455.
- [16] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, "SpanEdge: Towards unifying stream processing over central and near the-edge data centers," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2016, pp. 168-178.
- [17] D. Georgakopoulos, P. P. Jayaraman, M. Fazia, M. Villari, and R. Ranjan, "Internet of Things and edge cloud computing roadmap for manufacturing," *IEEE Cloud Comput.*, vol. 3, no. 4, pp. 66-73, Jul./Aug. 2016.
- [18] T. G. Rodrigues, K. Suto, H. Nishiyama, N. Kato and K. Temma, "Cloudlets Activation Scheme for Scalable Mobile Edge Computing with Transmission Power Control and Virtual Machine Migration," in *IEEE Transactions on Computers*, vol. 67, no. 9, pp. 1287-1300, 1 Sept. 2018, doi: 10.1109/TC.2018.2818144.

- [19] Liang, Bin, Mark A. Gregory, and Shuo Li. "Multi-access Edge Computing fundamentals, services, enablers and challenges: A complete survey." *Journal of Network and Computer Applications* 199 (2022): 103308.
- [20] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.
- [21] X. He, R. Jin, and H. Dai, "Peace: Privacy-preserving and cost-efficient task offloading for mobile-edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 1814–1824, Mar. 2020.
- [22] H. Liu, S. Hua, X. Zhuo, D. Chen, and X. Cheng, "Cooperative spectrum sharing of multiple primary users and multiple secondary users," *Digit. Commun. Netw.*, vol. 2, no. 4, pp. 191–195, 2016.
- [23] Kiran, Nahida, et al. "Joint resource allocation and computation offloading in mobile edge computing for SDN-based wireless networks." *Journal of Communications and Networks* 22.1 (2019): 1-11.
- [24] N. Piovesan, A. F. Gambin, M. Miozzo, M. Rossi, and P. Dini, "Energy sustainable paradigms and methods for future mobile networks: A survey," *Comput. Commun.*, vol. 119, pp. 101–117, Apr. 2018.
- [25] S. Bi and Y. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.
- [26] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 726–738, Sep. 2019.
- [27] Du, J., Jiang, C., Benslimane, A., Guo, S., and Ren, Y. (2022). SDN-based resource allocation in edge and cloud computing systems: An evolutionary stackelberg differential game approach. *IEEE/ACM Transactions on Networking*, 30(4), 1613-1628.
- [28] Wang, An, et al. "Software-defined networking enhanced edge computing: A network-centric survey." *Proceedings of the IEEE* 107.8 (2019): 1500-1519.

- [29] Ali, Jehad, et al. "A novel features prioritization mechanism for controllers in software-defined networking." *Comput. Mater. Contin* 69 (2021): 267-282.
- [30] Singh, Avtar, Navjot Kaur, and Harpreet Kaur. "Extensive performance analysis of OpenDayLight (ODL) and Open Network Operating System (ONOS) SDN controllers." *Microprocessors and Microsystems* 95 (2022): 104715.
- [31] Ali, Jehad, et al. "ESCALB: An effective slave controller allocation-based load balancing scheme for multi-domain SDN-enabled-IoT networks." *Journal of King Saud University-Computer and Information Sciences* 35.6 (2023): 101566.
- [32] Ali, Jehad, Byeong-hee Roh, and Seungwoon Lee. "QoS improvement with an optimum controller selection for software-defined networks." *Plos one* 14.5 (2019): e0217631.
- [33] Singh, Avtar, Navjot Kaur, and Harpreet Kaur. "An extensive vulnerability assessment and countermeasures in open network operating system software defined networking controller." *Concurrency and Computation: Practice and Experience* 34.15 (2022): e6978.
- [34] Thirupathi, V., et al. "A comprehensive review on sdn architecture, applications and major benefits of SDN." *International Journal of Advanced Science and Technology* 28.20 (2019): 607-614.
- [35] Foresta, Francesco, et al. "Improving OpenStack networking: Advantages and performance of native SDN integration." 2018 IEEE international conference on communications (ICC). IEEE, 2018.
- [36] Cabral, Marcos, Mauricio Silveira, and Ryan Urie. "SDN advantages for Ethernet-based control." *Schweitzer Eng. Lab., Inc., Rep., Pullman, WA, USA* (2019).
- [37] Singh, Avtar, Harpreet Kaur, and Navjot Kaur. "A novel DDoS detection and mitigation technique using hybrid machine learning model and redirect illegitimate traffic in SDN network." *Cluster Computing* (2023): 1-21.
- [38] Bera, Samaresh, et al. "Soft-WSN: Software-defined WSN management system for IoT applications." *IEEE Systems Journal* 12.3 (2016): 2074-2081.

- [39] Ren, Wei, et al. "BLLC: A batch-level update mechanism with low cost for SDN-IoT networks." *IEEE Internet of Things Journal* 6.1 (2018): 1210-1222.
- [40] Hong, Cheol-Ho, and Blesson Varghese. "Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms." *ACM Computing Surveys (CSUR)* 52.5 (2019): 1-37.
- [41] Keele, Staffs. "Guidelines for performing systematic literature reviews in software engineering." (2007).
- [42] BMJ (OPEN ACCESS) Page MJ, McKenzie JE, Bossuyt PM, Boutron I, Hoffmann TC, Mulrow CD, et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. *BMJ* 2021;372:n71. doi: 10.1136/bmj.n7.
- [43] Galluccio, Laura, et al. "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for Wireless SEnsor networks." 2015 IEEE conference on computer communications (INFOCOM). IEEE, 2015.
- [44] Hegazy, Tarek, and Moustafa Kassab. "Resource optimization using combined simulation and genetic algorithms." *Journal of construction engineering and management* 129.6 (2003): 698-705.
- [45] Heimonen, Klaus-Erik, and Josu A. Takala. "Resource optimization for sustainable competitive advantage in residential project development: empirical evidence from a medium-sized enterprise in Finland." *Strategic Optimization of Medium-Sized Enterprises in the Global Market*. IGI Global, 2019. 185-212.
- [46] Dar, Ab Rashid, and D. Ravindran. "Fog Computing Resource Optimization: A Review on Current Scenarios and Resource Management." *Baghdad Science Journal* 16.2 (2019).
- [47] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, A context sensitive offloading scheme for mobile cloud computing service," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, Jul. 2015, pp. 869-876.
- [48] Haghi Kashani, M., Rahmani, A. M., & Jafari Navimipour, N. (2020). Quality of service-aware approaches in fog computing. *International Journal of Communication Systems*, 33(8), e4340.

- [49] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795-2808, Oct. 2016.
- [50] Maritan, Catherine A., and Gwendolyn K. Lee. "Resource allocation and strategy." *Journal of Management* 43.8 (2017): 2411-2420.
- [51] Nayyer, Muhammad Ziad, et al. "LBRO: Load balancing for resource optimization in edge computing." *IEEE Access* 10 (2022): 97439-97449.
- [52] J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd, "Grid load balancing using intelligent agents," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 135–149, Jan. 2005, doi: 10.1016/j.future.2004.09.032.
- [53] J. Balasangameshwara and N. Raju, "A hybrid policy for fault tolerant load balancing in grid computing environments," *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 412–422, Jan. 2012, doi: 10.1016/j.jnca.2011.09.005.
- [54] Madej, Arkadiusz, et al. "Priority-based fair scheduling in edge computing." 2020 IEEE 4th international conference on fog and edge computing (ICFEC). IEEE, 2020.
- [55] Yuan, Hao, et al. "Online dispatching and fair scheduling of edge computing tasks: A learning-based approach." *IEEE Internet of Things Journal* 8.19 (2021): 14985-14998.
- [56] M. Haghi Kashani, A. M. Rahmani, and N. Jafari Navimipour, "Quality of service-aware approaches in fog computing," *Int. J. Commun. Syst.*, vol. 33, no. 8, 2020, doi: 10.1002/dac.4340.
- [57] A. A. Diro, H. T. Reda, and N. Chilamkurti, "Differential flow space allocation scheme in SDN based fog computing for IoT applications," *J. Ambient Intell. Humaniz. Comput.*, 2018, doi: 10.1007/s12652-017-0677-z.
- [58] H. Tran-Dang, S. Bhardwaj, T. Rahim, A. Musaddiq, and D.-S. Kim, "Reinforcement learning based resource management for fog computing environment: Literature review, challenges, and open issues," *J. Commun. Networks*, vol. 24, no. 1, pp. 83–98, 2022, doi: 10.23919/JCN.2021.000041.
- [59] J.-Y. Kwak¹, C. Cho, YongYoon Shin, and S. Yang, "IET Communications - 2020 - Kwak - IntelliTC intelligent inter-DC traffic controller for the Internet of everything.pdf," *IET Commun.*, 2019.

- [60] Madej, Arkadiusz, et al. "Priority-based fair scheduling in edge computing." 2020 IEEE 4th international conference on fog and edge computing (ICFEC). IEEE, 2020.
- [61] Yang, C., Liao, F., Lan, S., Wang, L., Shen, W., & Huang, G. Q. (2023). Flexible resource scheduling for software-defined cloud manufacturing with edge computing. *Engineering*, 22, 60-70.
- [62] Farooq, S., & Rathore, M. S. (2024). SDN-based fair packet scheduling for prioritising healthcare traffic in mobile edge computing: a comparative analysis with PQ and FCFS. *International Journal of Internet Protocol Technology*, 17(3), 155-166.
- [63] Murad, S. A., Azmi, Z. R. M., Muzahid, A. J. M., Bhuiyan, M. K. B., Saib, M., Rahimi, N., ... & Bairagi, A. K. (2024). SG-PBFS: Shortest gap-priority based fair scheduling technique for job scheduling in cloud environment. *Future Generation Computer Systems*, 150, 232-242.
- [64] Nain, A., Sheikh, S., & Shahid, M. (2025). An Efficient Load Distribution Approach for Optimizing Resources in SDN-Based Edge Computing Environment. *Concurrency and Computation: Practice and Experience*, 37(12-14), e70113.
- [65] Sun, H., Zhang, H., Ma, H., & Leung, V. C. (2025). Joint Scheduling, Computing, and Load Balancing for Time Sensitive Traffic in SDN-Enabled Space-Air-Ground Integrated 6 G Networks: A Federated Reinforcement Learning Approach. *IEEE Transactions on Mobile Computing*.
- [66] Tseng, C.W., Tseng, F.H., Yang, Y.T., Liu, C.C. and Chou, L.D., 2018. Task Scheduling for Edge Computing with Agile VNFs On-Demand Service Model toward 5G and Beyond. *Wireless Communications and Mobile Computing*, 2018(1), p.7802797.
- [67] Fang, X., Cai, Z., Tang, W., Luo, G., Luo, J., Bi, R. and Gao, H., 2019. Job scheduling to minimize total completion time on multiple edge servers. *IEEE Transactions on Network Science and Engineering*, 7(4), pp.2245-2255.
- [68] Gautam, S., & Malhotra, A. (2024). A Deadline-Aware Priority based Semi Greedy Task Scheduling Technique in Fog Computing. In *SCI* (pp. 218-228).
- [69] Zhu, G., Liu, T., Guo, Y., Xia, D., Lv, D. and Zhang, X., 2024, September. Research on Task Scheduling Methods in 5G Mobile Edge Computing

- Environments. In 2024 6th International Conference on Energy, Power and Grid (ICEPG) (pp. 1789-1794). IEEE.
- [70] Shiekh, S., Shahid, M., Sambare, M., Haidri, R.A. and Yadav, D.K., 2023. A load balanced hybrid heuristic for allocation of batch of tasks in cloud computing environment. *International Journal of Pervasive Computing and Communications*, 19(5), pp.756-781.
 - [71] Bu, C. and Wang, J., 2021. Computing tasks assignment optimization among edge computing servers via SDN. *Peer-to-Peer Networking and Applications*, 14, pp.1190-1206.
 - [72] Braunt, T.D., Siegel, H.J., Beck, N., Boloni, L.L., Maheswarans, M., Reuthert, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgeno, D. and Freund, R.F. (2000), "A comparison study of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", *ECE Technical Reports*, p. 19.
 - [73] Hao, Huizhen, Jie Zhang, and Qing Gu. "Optimal IoT service offloading with uncertainty in SDN-based mobile edge computing." *Mobile Networks and Applications* 27.6 (2022): 2318-2327.
 - [74] Xu, Xiaolong, et al. "Joint optimization of offloading utility and privacy for edge computing enabled IoT." *IEEE Internet of Things Journal* 7.4 (2019): 2622-2629.
 - [75] Wang, S., Zhou, A., Bao, R., Chou, W., & Yau, S. S. (2018). Towards green service composition approach in the cloud. *IEEE Transactions on Services Computing*, 14(4), 1238-1250.
 - [76] Abdollahi, S., Deldari, A., Asadi, H., Montazerolghaem, A., & Mazinani, S. M. (2021). Flow-aware forwarding in SDN datacenters using a knapsack-PSO-based solution. *IEEE Transactions on Network and Service Management*, 18(3), 2902-2914.
 - [77] Negi, G., Kumar, A., Pant, S., & Ram, M. (2021). GWO: a review and applications. *International Journal of System Assurance Engineering and Management*, 12, 1-8.
 - [78] Keele, Staffs. "Guidelines for performing systematic literature reviews in software engineering." (2007).

- [79] BMJ (OPEN ACCESS) Page MJ, McKenzie JE, Bossuyt PM, Boutron I, Hoffmann TC, Mulrow CD, et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. *BMJ* 2021;372:n71. doi: 10.1136/bmj.n7.
- [80] Nain, A., Sheikh, S., Shahid, M. and Malik, R., 2024. Resource optimization in edge and SDN-based edge computing: a comprehensive study. *Cluster Computing*, 27(5), pp.5517-5545.
- [81] A. Filali, A. Abouaomar, S. Cherkaoui, A. Kobbane and M. Guizani, "Multi-Access Edge Computing: A Survey," in *IEEE Access*, vol. 8, pp. 197017-197046, 2020, doi: 10.1109/ACCESS.2020.3034136.
- [82] C. Jiang, X. Cheng, H. Gao, X. Zhou, and J. Wan, "Toward computation offloading in edge computing: A survey," *IEEE Access*, vol. 7, pp. 131543–131558, 2019.
- [83] A. Islam, A. Debnath, M. Ghose, and S. Chakraborty, "A survey on task offloading in multi-access edge computing," *J. Syst. Archit.*, vol. 118, Sep. 2021, Art. no. 102225.
- [84] Q. Luo, S. Hu, C. Li, G. Li and W. Shi, "Resource Scheduling in Edge Computing: A Survey," in *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2131-2165, Fourthquarter 2021, doi: 10.1109/COMST.2021.3106401.
- [85] S. Bi and Y. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.
- [86] A.Amjad, F. Rabby, S. Sadia, M. Patwary, and E. Benkhelifa, Cognitive edge computing based resource allocation framework for Internet of Things, in *Proc. 2nd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, May 2017, pp. 194200.
- [87] Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Commun. Lett.*, vol. 6, no. 3, pp. 398401, Jun. 2017.
- [88] Li, Shichao; Zhang, Ning; Lin, Siyu; Kong, Linghe; Katangur, Ajay; Khan, Muhammad Khurram; Ni, Minming; Zhu, Gang (2018). Joint Admission

Control and Resource Allocation in Edge Computing for Internet of Things. *IEEE Network*, 32(1), 72–79. doi:10.1109/MNET.2018.1700163.

- [89] Xu, Jinlai; Palanisamy, Balaji; Ludwig, Heiko; Wang, Qingyang (2017). [IEEE 2017 IEEE International Conference on Edge Computing (EDGE) - Honolulu, HI, USA (2017.6.25-2017.6.30)] 2017 IEEE International Conference on Edge Computing (EDGE) - Zenith: Utility-Aware Resource Allocation for Edge Computing. , (), 47–54. doi:10.1109/IEEE.EDGE.2017.15).
- [90] P. Rimal, D. P. Van, and M. Maier, "Mobile edge computing empowered fiber-wireless access networks in the 5G era," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 192200, Feb. 2017.
- [91] P. Rimal, D. P. Van, and M. Maier, "Cloudlet enhanced fiber-wireless access networks for mobile-edge computing," *IEEE Trans. Wireless Commun.*, vol. 16, no. 6, pp. 36013618, Jun. 2017.
- [92] M. Chen, Y. Hao, Y. Li, C.-F. Lai, and D. Wu, "On the computation offloading at ad hoc cloudlet: Architecture and service modes," *IEEE Commun. Mag.*, vol. 53, no. 6, pp. 1824, Jun. 2015.
- [93] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 1423, Oct./Dec. 2009.
- [94] Qin, Zhenquan; Qiu, Xueyan; Ye, Jin; Wang, Lei (2020). User-Edge Collaborative Resource Allocation and Offloading Strategy in Edge Computing. *Wireless Communications and Mobile Computing*, 2020(), 1–12. doi:10.1155/2020/88671.
- [95] Liu, X., Qin, Z., & Gao, Y. (2019). Resource Allocation for Edge Computing in IoT Networks via Reinforcement Learning. *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. doi:10.1109/icc.2019.8761385.
- [96] R. Beraldi, A. Mtibaa, and H. Alnuweiri, "Cooperative load balancing scheme for edge computing resources," in *Proc. 2nd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, May 2017, pp. 94100.
- [97] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, and M. Satyanarayanan, "Adaptive VM handoff across cloudlets," *Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-15-113*, Jun. 2015.

- [98] R. Agarwal and A. Nayak, "DRAP: A decentralized public resourced cloudlet for ad-hoc networks," in Proc. IEEE 4th Int. Conf. Cloud Netw. (CloudNet), Oct. 2015, pp. 309314.
- [99] L. Tang, X. Chen, and S. He, "When social network meets mobile cloud: A social group utility approach for optimizing computation offloading in cloudlet," IEEE Access, vol. 4, pp. 58685879, 2016.
- [100] Yaozhong Song, Stephen S. Yau. (2017). An Approach to QoS-based Task Distribution in Edge Computing Networks for IoT Applications. IEEE 1st International Conference on Edge Computing.
- [101] Tham, Chen-Khong; Chattopadhyay, Rajarshi (2017). [IEEE 2017 IEEE 18th International Symposium on " World of Wireless, Mobile and Multimedia Networks (WoWMoM) - Macau, China (2017.6.12-2017.6.15)] 2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM) - A load balancing scheme for sensing and analytics on a mobile edge computing network. , (), 1-9. doi:10.1109/WoWMoM.2017.7974307.
- [102] Babou, Cheikh Saliou Mbacke; Fall, Doudou; Kashihara, Shigeru; Tae-naka, Yuzo; Bhuyan, Monowar H.; Niang, Ibrahima; Kadobayashi, Youki (2020). Hierarchical Load Balancing and Clustering Technique for Home Edge Computing. IEEE Access, 8(), 127593–127607. doi:10.1109/ACCESS.2020.3007944.
- [103] Kadhim, Ahmed Jawad, and Jaber Ibrahim Naser. "Proactive load balancing mechanism for fog computing supported by parked vehicles in IoV-SDN." China Communications 18.2 (2021): 271-289.
- [104] Zhang, Wei-Zhe, et al. "Secure and optimized load balancing for multitier IoT and edge-cloud computing systems." IEEE IoT Journal 8.10 (2020): 8119-8132.
- [105] J. L. J. Laredo, F. Guinand, D. Olivier, and P. Bouvry, "Load balancing at the edge of chaos: How self-organized criticality can lead to energy- efficient computing," IEEE Trans. Parallel Distrib. Syst., vol. 28, no. 2, pp. 517529, Feb. 2017.

- [106] L. Li, Y. Li, and R. Hou, "A novel mobile edge computing-based architecture for future cellular vehicular networks," in Proc. IEEE WirelessCommun. Netw. Conf. (WCNC), Mar. 2017, pp. 16.
- [107] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," IEEE J. Sel.AreasCommun., vol. 34, no. 12, pp. 35903605, Dec. 2016.
- [108] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task ofloading scheduling and transmit power allocation for mobile-edge computing systems," in Proc.IEEE Wireless Commun. Netw. Conf. (WCNC), Mar. 2017, pp. 16.
- [109] S. Bohez, J. De Turck, T. Verbelen, P. Simoens, and B. Dhoedt, "Mobile, collaborative augmented reality using cloudlets," in Proc. Int. Conf. MOBILE Wireless MiddleWARE, Operating Syst., Appl., Nov. 2013, pp. 4554.
- [110] S. Moon and Y. Lim, "Task migration with partitioning for load balancing in collaborative edge computing," Appl. Sci., vol. 12, no. 3, p. 1168, 2022. [Online]. Available: <https://www.mdpi.com/2076-3417/12/3/1168>.
- [111] Rahman, A., Chakraborty, C., Anwar, A., Karim, M.R., Islam, M.J., Kundu, D., Rahman, Z., Band, S.S.: SDN-IoT empowered intelligent framework for industry 4.0 applications during COVID-19 pandemic. Cluster Comput. 4, (2021). <https://doi.org/10.1007/s10586-021-03367-4>.
- [112] Isyaku, B., MohdZahid, M.S., BteKamat, M., Abu Bakar, K., Ghaleb, F.A.: Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey. Futur. Internet. 12, 147 (2020). <https://doi.org/10.3390/fi12090147>.
- [113] Meneghello, F., Calore, M., Zucchetto, D., Polese, M., Zanella, A.: IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices. IEEE Internet Things J. 6, 8182-8201 (2019). <https://doi.org/10.1109/JIOT.2019.2935189>.
- [114] Yurchenko, M., Cody, P., Coplan, A., Kennedy, R., Wood, T., Ramakrishnan, K.K.: OpenNetVM. 1-2 (2018). <https://doi.org/10.1145/3185467.3190786>.
- [115] aksymyuk, T., Brych, M., Dumych, S., Satria, D., Jo, M.: An IoT based monitoring framework for software defined 5G mobile networks. Proc. 11th

- Int. Conf. Ubiquitous Inf. Manag. Commun. IMCOM 2017. 7-10 (2017).
<https://doi.org/10.1145/3022227.3022331>.
- [116] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Arbor, A., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Arbor, A., Invernizzi, L., Kallitsis, M., Network, M., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., Zhou, Y., Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., Zhou, Y.: Understanding the Mirai Botnet. *USENIX Secur.* 1093-1110 (2017).
<https://doi.org/10.1016/j.religion.2008.12.001>.
 - [117] Du, P., Putra, P., Yamamoto, S., Nakao, A.: A context-aware IoT architecture through software-defined data plane. *Proc. - 2016 IEEE Reg. 10 Symp. TENSYP 2016.* 315-320 (2016).
<https://doi.org/10.1109/TENCONSpring.2016.7519425>.
 - [118] Jazaeri, Seyedeh Shabnam, et al. "Composition of caching and classification in edge computing based on quality optimization for SDN-based IoT healthcare solutions." *The Journal of Supercomputing* (2023): 1-51.
 - [119] Singh, Jagdeep, et al. "Energy-Efficient and Secure Load Balancing Technique for SDN-Enabled Fog Computing." *Sustainability* 14.19 (2022): 12951.
 - [120] Chen, Yu-Jia, et al. "SDN-enabled traffic-aware load balancing for M2M networks." *IEEE Internet of Things Journal* 5.3 (2018): 1797-1806.
 - [121] Zhang, Jie, et al. "Task offloading in vehicular edge computing networks: A load-balancing solution." *IEEE Transactions on Vehicular Technology* 69.2 (2019): 2092-2104.
 - [122] Khiat, A., Haddadi, M., & BAHNES, N. (2024). Genetic-Based Algorithm for Task Scheduling in Fog-Cloud Environment. *Journal of Network and Systems Management*, 32(1).
 - [123] Zhou, X., Yang, J., Li, Y., Li, S. and Su, Z., 2024. Deep reinforcement learningbased resource scheduling for energy optimization and load balancing in SDNdriven edge computing. *Computer Communications*, 226, p.107925.
 - [124] Nain, A., Sheikh, S., Malik, R. and Mushtaq, S.U., 2023, November. Hybrid Metaheuristic Based Resource Allocation Approach in SDN-Based Edge

- Computing Environment. In 2023 2nd International Conference on Futuristic Technologies (INCOFT) (pp. 1-5). IEEE.
- [125] Lakhan, A., Mohammed, M. A., Obaid, O. I., Chakraborty, C., Abdulka-reem, K. H. and Kadry, S. (2022). Efficient deep-reinforcement learning aware resource allocation in SDN-enabled fog paradigm. *Automated Software Engineering*, 29,1-25.
 - [126] Ali, J., Jhaveri, R. H., Alswailim, M., and Roh, B. H. (2023). ESCALB: An effective slave controller allocation-based load balancing scheme for multi-domain SDN-enabled-IoT networks. *Journal of King Saud University-Computer and Information Sciences*, 35(6), 101566.
 - [127] Du, J., Jiang, C., Benslimane, A., Guo, S. and Ren, Y., 2022. SDN-based resource allocation in edge and cloud computing systems: An evolutionary Stackelberg differential game approach. *IEEE/ACM Transactions on Networking*, 30(4), pp.1613-1628.
 - [128] Liu, Y., Tan, C., Wang, K., & Chen, W. (2024). Hybrid Task Offloading and Resource Optimization in Vehicular Edge Computing Networks. *IEEE Wireless Communications Letters*.
 - [129] Tian, S., Xiang, S., Zhou, Z., Dai, H., Yu, E. and Deng, Q., 2025. Task Offloading and Resource Allocation Based on Reinforcement Learning and Load Balancing in Vehicular Networking. *IEEE Transactions on Consumer Electronics*.
 - [130] Zhu, S.F., Liu, C.T., Zhu, H., Qiao, R., Chen, H. and Zhang, H., 2025. Multi-objective optimization task offloading decision for intelligent transportation system in cloud edge collaborative computing scenario. *Wireless Networks*, pp.1-20.
 - [131] Darabkh, K.A., Al-Mistarihi, M.F. and Odat, B.A., 2025. Leveraging fog computing and software-defined networking for a novel velocity-aware routing protocol with election and handover thresholds in VANETs. *The Journal of Supercomputing*, 81(2), pp.1-37.
 - [132] Zhao, L., Feng, Y., Hawbani, A., Xu, L., Liu, Z. and Bi, Y., 2025. Optimized Resource Allocation in Vehicle Edge Computing through Platoon Collaboration. *IEEE Internet of Things Journal*.

- [133] Chai, Y., Zeng, X.J., Chen, Q. and Cheng, L., 2025. Stackelberg Game-Based Joint Computing Resource Allocation and Task Offloading Method in Edge Computing. *IEEE Transactions on Vehicular Technology*.
- [134] Wu, G., Wang, H., Zhang, H., Shen, Y., Shen, S. and Yu, S., 2024. Mean-field game-based task-offloaded load balance for industrial mobile edge computing systems using software-defined networking. *IEEE Transactions on Mobile Computing*.
- [135] AL-Bakhrani, A.A., Li, M., Obaidat, M.S. and Amran, G.A., 2025. MOALF-UAV-MEC: Adaptive Multi-Objective Optimization for UAV-Assisted Mobile Edge Computing in Dynamic IoT Environments. *IEEE Internet of Things Journal*.
- [136] Jasim, A.M. and Al-Raweshidy, H., 2024. An adaptive sdn-based load balancing method for edge/fog-based real-time healthcare systems. *IEEE Systems Journal*.
- [137] Akyildiz, Ian F., Shih-Chun Lin, and Pu Wang. "Wireless software-defined networks (W-SDNs) and network function virtualization (NFV) for 5G cellular systems: An overview and qualitative evaluation." *Computer Networks* 93 (2015): 66-79.
- [138] Peng, H., Ye, Q., and Shen, X. S. (2019). SDN-based resource management for autonomous vehicular networks: A multi-access edge computing approach. *IEEE Wireless Communications*, 26(4), 156-162.
- [139] R. F. Moyano, D. Fernandez, L. Bellido, N. Merayo, J. C. Aguado, and I. de Miguel, "Nfv-based qos provision for software defined optical access and residential networks," in *Quality of Service (IWQoS)*, 2017 IEEE/ACM 25th International Symposium on. IEEE, 2017, pp. 1–5.
- [140] Yao, J.; Han, Z.; Sohail, M.; Wang, L. A Robust Security Architecture for SDN-Based 5G Networks. *FutureInternet* 2019, 11, 85.
- [141] Mitsis, G.; Apostolopoulos, P.A.; Tsiropoulou, E.E.; Papavassiliou, S. Intelligent Dynamic Data Offloading in a Competitive Mobile Edge Computing Market. *Future Internet* 2019, 11, 118.
- [142] Leivadreas, A.; Kesidis, G.; Ibnkahla, M.; Lambadaris, I. VNF Placement Optimization at the Edge and Cloud. *Future Internet* 2019, 11, 69.

- [143] Chen, Y., Ma, J., Huang, L., and Zhang, X. (2021). Energy-aware resource allocation and task scheduling in edge computing systems. *IEEE Transactions on Sustainable Computing*, 6(1), 83-95.
- [144] Zhang, H., Yang, W., Xiang, Y., and Xiang, Z. (2020). Dynamic resource allocation for edge computing using software-defined networking. *Future Generation Computer Systems*, 110, 362-372.
- [145] Wang, S., Zhang, L., Wang, Y., and Zhu, X. (2021). Resource allocation scheme for multi-access edge computing using software-defined networking. *Journal of Network and Computer Applications*, 176, 102938.
- [146] Balasubramanian, Venkatraman, et al. "Intelligent resource management at the edge for ubiquitous IoT: An SDN-based federated learning approach." *IEEE network* 35.5 (2021): 114-121
- [147] Skondras, Emmanouil, et al. "QoS-aware scheduling in LTE-A networks with SDN control." 2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA). IEEE, 2016.
- [148] Jian Zhang, Qimei Cui, Xuefei Zhang, Wei Ni, Xincheng Lyu, Miao Pan, Xiaofeng Tao et al. "Online Optimization of Energy-Efficient User Association and Workload Offloading for Mobile Edge Computing". *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY*, VOL. 71, NO. 2, FEBRUARY 2022.
- [149] Wei Guo, V. Mahendran, Sridhar Radhakrishnan et al. "Achieving Throughput Fairness in Smart Grid Using SDN-Based Flow Aggregation and Scheduling". 2016 Ninth IEEE International Workshop on Selected Topics in Mobile and Wireless Computing.
- [150] Feng, Tao, Jun Bi, and Ke Wang. "Allocation and scheduling of network resource for multiple control applications in sdn." *China Communications* 12.6 (2015): 85-95.
- [151] Abhilasha Sharma, Lalit Kumar Awasthi et al. "Pr-CAI: Priority based-Context Aware Information scheduling for SDN-based vehicular network". *Computer Networks* 193 (2021) 108097.
- [152] Chen, Kang, et al. "Improving integrated LTE-WiFi network performance with SDN based flow scheduling." 2018 27th international conference on computer communication and networks (ICCCN). IEEE, 2018.

- [153] Tayyaba, Sahrish Khan, et al. "Software-defined network (sdn) based internet of things (iot) a road ahead." Proceedings of the international conference on future networks and distributed systems. 2017.
- [154] Kalkan, Kubra, and Sherali Zeadally. "Securing internet of things with software defined networking." IEEE Communications Magazine 56.9 (2017): 186-192.
- [155] Alvizu, Rodolfo, et al. "Comprehensive survey on T-SDN: Software-defined networking for transport networks." IEEE Communications Surveys & Tutorials 19.4 (2017): 2232-2283.
- [156] Bera, Samaresh, Sudip Misra, and Athanasios V. Vasilakos. "Software-defined networking for internet of things: A survey." IEEE Internet of Things Journal 4.6 (2017): 1994-2008.
- [157] Carvalho, Gonçalo, et al. "Edge computing: current trends, research challenges and future directions." Computing 103 (2021): 993-1023.
- [158] Foukas, Xenofon, et al. "Network slicing in 5G: Survey and challenges." IEEE communications magazine 55.5 (2017): 94-100.

List of Publications

1. **Ajay Nain, Sophiya Sheikh, Mohammad Shahid, Rohit Malik** "Resource optimization in edge and SDN-based edge computing: a comprehensive study." *Cluster Computing* 27, no. 5 (2024): 5517-5545.(SCI)Q1
2. **Ajay Nain, Sophiya Sheikh, Mohammad Shahid** "An Efficient Load Distribution Approach for Optimizing Resources in SDN-Based Edge Computing Environment." *Concurrency and Computation: Practice and Experience* 37, no. 12-14 (2025): e70113.(SCI)Q1
3. **Ajay Nain, Sophiya Sheikh, Mohammad Shahid, Sheikh Umar Mushtaq** Priority Aware Task Scheduling Technique for Efficient Resource Utilization in SDN-based Edge Computing (Communicated).
4. **Ajay Nain, Sophiya Sheikh, Mohammad Shahid, Sheikh Umar Mushtaq** Adaptive Joint Allocation for Dynamic Fair Scheduling in SDN-Based Edge Computing (Communicated).

Papers Presented in Conferences

1. **Ajay Nain, Sophiya Sheikh, Rohit** Software-defined network: An emerging solution for IoT-CC-edge paradigm—An impeccable study(RACS 2022).
2. **Ajay Nain, Sophiya Sheikh, Rohit, Sheikh Umar Mushtaq** Hybrid Metaheuristic Based Resource Allocation Approach in SDN-Based Edge Computing Environment(INCOFT) 2023.

Patents

1. Designing an efficient framework for big data security in Fog-enabled IoT environment.-202411068980
2. Optimizing SDN-Based Edge Computing with Deep Belief Networks for Predictive Analytics and Intrusion Detection.