

**CONTAINER MIGRATION APPROACH FOR IOT BASED  
EDGE-CLOUD SYSTEMS**

**(NESNELERIN INTERNETİ BAZLI UÇ BULUT SİSTEMLERİ İÇİN  
KONTEYNER TAŞIMA YAKLAŞIMI)**

by

**MEHMET BERKAY PALA, B.S.**

**Thesis**

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

**MASTER OF SCIENCE**

in

**INTELLIGENT SYSTEMS ENGINEERING**

in the

**GRADUATE SCHOOL OF SCIENCE AND ENGINEERING**

of

**GALATASARAY UNIVERSITY**

**NOV 2025**

## ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor, Assist. Prof. Dr. Ozgun PINARER, for his guidance and support throughout this research. I would like to express my deepest gratitude to my wife, Kevser, for her inexpressible valuable support, love and for the profound impact her presence has had on my life. I am also sincerely thankful to my family for their continuous encouragement, love, and belief in me throughout my life.

**Mehmet Berkay Pala**

*November, 2025*

# TABLE OF CONTENTS

|                                                           |      |
|-----------------------------------------------------------|------|
| ACKNOWLEDGEMENTS . . . . .                                | iii  |
| TABLE OF CONTENTS . . . . .                               | iv   |
| LIST OF FIGURES . . . . .                                 | vii  |
| LIST OF TABLES . . . . .                                  | ix   |
| ABSTRACT . . . . .                                        | x    |
| RÉSUMÉ . . . . .                                          | xi   |
| ÖZET . . . . .                                            | xii  |
| LIST OF ABBREVIATIONS . . . . .                           | xiii |
| 1 INTRODUCTION . . . . .                                  | 1    |
| 2 LITERATURE REVIEW . . . . .                             | 5    |
| 2.1 Container Migration Techniques . . . . .              | 5    |
| 2.2 Task Scheduling and Orchestration . . . . .           | 12   |
| 2.3 Edge Computing Frameworks and Architectures . . . . . | 18   |
| 3 PROPOSED APPROACH . . . . .                             | 28   |
| 3.1 Overview . . . . .                                    | 28   |
| 3.2 Container-based Edge & Central IoT . . . . .          | 29   |
| 3.2.1 Contanierization . . . . .                          | 29   |
| 3.2.2 Edge & Central Kubernetes Architecture . . . . .    | 30   |
| 3.2.3 CRIU and Live Migration of Workloads . . . . .      | 31   |
| 3.3 IoT Service Migration Scenarios . . . . .             | 34   |
| 3.4 Service-based IoT Communication . . . . .             | 35   |
| 3.5 Metrics & Metric Management . . . . .                 | 37   |
| 3.6 Fuzzy Logic based Migration Decision Making . . . . . | 39   |

|         |                                                               |    |
|---------|---------------------------------------------------------------|----|
| 3.6.1   | Multidimensional Input Metrics . . . . .                      | 42 |
| 3.6.2   | Fuzzification . . . . .                                       | 44 |
| 3.6.3   | Rule Base Structure . . . . .                                 | 45 |
| 3.6.4   | Inference . . . . .                                           | 46 |
| 3.6.5   | Defuzzification . . . . .                                     | 46 |
| 3.6.6   | Continuous Evaluation . . . . .                               | 47 |
| 3.7     | Migration Orchestration . . . . .                             | 47 |
| 3.7.1   | Decision Making & Migration Orchestration Workflow            | 48 |
| 3.7.1.1 | Fuzzy Logic System Module & Fuzzy Logic Algorithms . . . . .  | 49 |
| 3.7.1.2 | Metric Collection & Aggregation Module .                      | 52 |
| 3.7.1.3 | CRIU Integration Module . . . . .                             | 54 |
| 3.7.1.4 | Worker Node State Tracking Module . . .                       | 55 |
| 3.7.1.5 | Migration Manager Module . . . . .                            | 56 |
| 3.7.1.6 | Main Implementation Module . . . . .                          | 58 |
| 4       | EXPERIMENTS . . . . .                                         | 59 |
| 4.1     | Test Environment . . . . .                                    | 59 |
| 4.2     | Advanced Simulation . . . . .                                 | 60 |
| 4.3     | Migration Scenarios . . . . .                                 | 61 |
| 5       | DISCUSSION & CONCLUSION . . . . .                             | 65 |
| 5.1     | Mamdani & Sugeno-based Migration Simulation based Discussions | 65 |
| 5.1.1   | Basic Simulation . . . . .                                    | 65 |
| 5.1.2   | Advanced Simulation . . . . .                                 | 66 |
| 5.2     | Conclusion . . . . .                                          | 71 |
| 5.3     | Future Studies . . . . .                                      | 73 |
| 5.3.1   | Physical Test Environment Study . . . . .                     | 74 |
|         | REFERENCES . . . . .                                          | 77 |
|         | APPENDICES . . . . .                                          | 81 |
|         | Appendix . A . . . . .                                        | 81 |

Publications . . . . . 81



## LIST OF FIGURES

|                                                                                                 |    |
|-------------------------------------------------------------------------------------------------|----|
| Figure 1.1 Illustration of Container Workload Migration in Edge based IoT Environment . . . . . | 4  |
| Figure 3.1 High-level Architecture Diagram . . . . .                                            | 29 |
| Figure 3.2 CRIU based Checkpoint Flow . . . . .                                                 | 33 |
| Figure 3.3 Service based Architecture & Service Registry Diagram . . . . .                      | 37 |
| Figure 3.4 Fuzzy Inference System . . . . .                                                     | 40 |
| Figure 3.5 Structure of Decision Making & Migration Orchestration System . .                    | 50 |
| Figure 3.6 Fuzzy Logic Module . . . . .                                                         | 51 |
| Figure 3.7 Metric Collection & Aggregation Module . . . . .                                     | 54 |
| Figure 3.8 CRIU Integration Module . . . . .                                                    | 55 |
| Figure 3.9 Worker Node State Tracking Module . . . . .                                          | 56 |
| Figure 3.10 Migration Manager Module . . . . .                                                  | 57 |
| Figure 4.1 Python based Simulation - Input Variables . . . . .                                  | 61 |
| Figure 4.2 Python based Simulation - Membership Functions . . . . .                             | 61 |
| Figure 4.3 Migration Scenarios . . . . .                                                        | 63 |
| Figure 4.4 Python based Simulation Logic . . . . .                                              | 64 |
| Figure 5.1 Fuzzy Rules & Membership Functions . . . . .                                         | 65 |

|                                                                                                       |    |
|-------------------------------------------------------------------------------------------------------|----|
| Figure 5.2 Heatmap of Migration Events Across Nodes & Migration Decision<br>Over Time Trend . . . . . | 66 |
| Figure 5.3 New Worker Deployment - Box Plot . . . . .                                                 | 67 |
| Figure 5.4 New Worker Deployment - Migration Plots . . . . .                                          | 68 |
| Figure 5.5 Worker Failover - Migration Plots . . . . .                                                | 69 |
| Figure 5.6 Worker Failover - Migration Rate Plot . . . . .                                            | 70 |
| Figure 5.7 Mamdani vs. Sugeno - Score Distribution . . . . .                                          | 70 |
| Figure 5.8 Mamdani vs. Sugeno - Total Migration Decisions . . . . .                                   | 71 |
| Figure 5.9 Offers of the proposed approach . . . . .                                                  | 73 |
| Figure 5.10Future study . . . . .                                                                     | 74 |

LIST OF TABLES

Table 2.1 Overview of Existing Studies on Container Migration Techniques . . 11

Table 2.2 Overview of Existing Studies on Task Scheduling and Orchestration . 17

Table 2.3 Overview of Existing Studies on Edge Computing Frameworks and  
Architectures . . . . . 25

Table 2.4 Overview of existing studies . . . . . 27

## ABSTRACT

The increasing demand for edge computing and the rise of cloud systems have led to the need for efficient resource management and workload distribution across edge and central cloud environments. Containerization has emerged as a popular solution for application deployment and management due to its lightweight and portable nature. However, migrating containers between edge and cloud environments horizontally and vertically poses challenges in terms of network latency, resource constraints, and service availability. This research project aims to develop a container migration approach specifically tailored for edge-cloud systems. The objective is to design and implement a mechanism that enables seamless and efficient container migration between edge devices and cloud infrastructure. By leveraging the advantages of both edge and cloud computing, it will be possible to allocate resources dynamically based on workload demands and optimize the overall system performance. Key aspects of this project include investigating the challenges and requirements of container migration in edge-cloud systems, designing an efficient migration mechanism that considers network latency, resource availability, and service continuity, evaluating the performance and reliability of the migration approach under various scenarios and workloads, analyzing the cost-effectiveness and resource utilization of the proposed solution compared to traditional deployment models, and developing strategies for load balancing, fault tolerance, and workload optimization to enhance the overall system efficiency. This project aims to provide insights into the feasibility and benefits of container migration in edge-cloud systems. The findings will contribute to the development of more efficient resource management strategies and enable the seamless deployment and migration of containers across edge and cloud environments.

**Keywords :** Edge-Cloud Systems, Containerization, Resource Management, Workload Distribution, Seamless Migration

## RÉSUMÉ

L'augmentation de la demande pour le calcul en périphérie est engendré le besoin d'une gestion efficace des ressources et de la distribution de la charge de travail à travers les environnements périphériques. La conteneurisation est apparue comme une solution populaire pour le déploiement et la gestion des applications en raison de sa légèreté et de sa portabilité. Cependant, la migration des conteneurs entre les environnements périphériques et les nuages, pose des défis en termes de latence réseau. Ce projet de recherche vise à développer une approche de migration de conteneurs spécifiquement adaptée aux systèmes Edge-Cloud. L'objectif est de concevoir et de mettre en œuvre un mécanisme permettant une migration de conteneurs fluide et efficace entre les dispositifs périphériques et l'infrastructure cloud. En exploitant les avantages du calcul en périphérie et du cloud, il sera possible d'allouer dynamiquement des ressources en fonction des demandes de charge de travail et d'optimiser les performances globales du système. Les aspects clés de ce projet comprennent l'exploration des défis et des exigences de la migration des conteneurs dans les systèmes Edge-Cloud, la conception d'un mécanisme de migration efficace qui prend en compte la latence réseau, la disponibilité des ressources et la continuité des services, l'évaluation des performances et de la fiabilité de l'approche de migration dans divers scénarios et charges de travail, l'analyse de la rentabilité et de l'utilisation des ressources de la solution proposée par rapport aux modèles de déploiement traditionnels, et le développement de stratégies d'équilibrage de charge, de tolérance aux pannes et d'optimisation de la charge de travail pour améliorer l'efficacité globale du système. Ce projet vise à fournir des informations sur la faisabilité et les avantages de la migration des conteneurs dans les systèmes Edge-Cloud. Les résultats contribueront au développement de stratégies de gestion des ressources plus efficaces et permettront le déploiement et la migration transparents des conteneurs à travers les environnements Edge et Cloud.

**Mots Clés :** Systèmes Edge-Cloud, Conteneurisation, Gestion des ressources, Distribution de la charge de travail, Migration transparente

## ÖZET

Uç bilişimin talebinin artması ve bulut sistemlerinin yükselişi, uç ve merkezi bulut ortamları arasında verimli kaynak yönetimi ve iş yükü dağıtımının gerekliliğini ortaya koymuştur. Esnek, taşınabilir ve hafif doğası nedeniyle konteynerleştirme, uygulama dağıtımı ve yönetimi için popüler bir çözüm olarak ortaya çıkmıştır. Ancak, konteynerlerin uç ve bulut ortamları arasında yatay ve dikey olarak taşınması, ağ gecikmesi, kaynak kısıtları ve hizmet sürekliliği gibi zorluklarla karşılaşmaktadır. Bu proje, uç-bulut sistemleri için özel olarak tasarlanmış yatayda ve dikeyde konteyner taşımayı sağlayacak yaklaşımı geliştirmeyi amaçlamaktadır. Hedef, uç cihazlar ile bulut altyapısı arasında sorunsuz ve etkili bir şekilde konteyner taşımaları sağlayacak bir mekanizma tasarlamak ve uygulamaktır. Uç ve bulut bilişimin avantajlarını kullanarak, iş yükü taleplerine göre kaynakları dinamik olarak tahsis etmek ve genel sistem performansını optimize etmek mümkün olacaktır. Bu projenin temel yönleri, uç-bulut sistemlerinde konteyner taşımanın zorluklarını ve gereksinimlerini araştırmak, ağ gecikmesi, kaynak erişilebilirliği ve hizmet sürekliliğini dikkate alan verimli bir taşıma mekanizması tasarlamak, çeşitli senaryo ve iş yükleri altında taşıma yaklaşımının performansını ve güvenilirliğini değerlendirmek, önerilen çözümün geleneksel dağıtım modelleri ile karşılaştırıldığında maliyet etkinliğini ve kaynak kullanımını analiz etmek ve genel sistem verimliliğini artırmak için yük dengeleme, hata tolere edicilik ve iş yükü optimizasyonu stratejileri geliştirmektir. Bu proje, uç-bulut sistemlerinde konteyner taşımanın olanaklılığı ve faydaları hakkında bilgi sağlamayı amaçlamaktadır. Bulgular, daha verimli kaynak yönetimi stratejilerinin geliştirilmesine ve uç ve bulut ortamları arasında konteynerlerin sorunsuz dağıtım ve taşınmasının sağlanmasına katkıda bulunacaktır.

**Anahtar Kelimeler :** Uç-Bulut Sistemleri, Konteynerleştirme, Kaynak Yönetimi, İş Yükü Dağıtımı, Sorunsuz Göç

## LIST OF ABBREVIATIONS

|      |                                   |
|------|-----------------------------------|
| API  | Application Programming Interface |
| CPU  | Central Processing Unit           |
| CRIU | Checkpoint/Restore In Userspace   |
| FIS  | Fuzzy Inference System            |
| HTTP | HyperText Transfer Protocol       |
| I/O  | Input/Output                      |
| IoT  | Internet of Things                |
| IIoT | Industrial Internet of things     |
| IT   | Information Technology            |
| K8S  | Kubernetes                        |
| QoS  | Quality of Service                |
| SBA  | Service based Architecture        |
| SLA  | Service Level Agreement           |
| TPS  | Transaction per Second            |
| VM   | Virtual Machine                   |

## 1 INTRODUCTION

In recent years, the convergence of various technological advancements such as connectivity, hardware, software and data analytics has led to the exponential growth of Internet of Things (IoT) devices, revolutionizing the landscape of computing paradigms. Particularly, the emergence of Edge computing has garnered significant attention as a transformative approach to processing and managing data generated by IoT devices (Gand et al., 2021). Edge computing, by bringing computation and data storage closer to the source of data generation, offers several advantages over traditional cloud-centric architectures, including reduced latency, improved bandwidth utilization, and possibility to achieve enhanced privacy and security. This shift towards Edge computing has been driven by the increasing demand for real-time analytics, low-latency applications, and efficient resource utilization across diverse domains such as industrial automation, healthcare, smart cities, and transportation systems.

In tandem with the proliferation of IoT devices and the adoption of Edge computing, the management of software applications in decentralized and dynamic Edge environments has become increasingly complex. Traditional methods of application deployment and management, often reliant on monolithic architectures or virtual machines, struggle to meet the demands of Edge computing due to challenges such as resource constraints, network variability, and the need for rapid scalability. In this context, containerization has emerged as a promising solution to address these challenges and streamline the deployment and management of applications in Edge environments.

The rapid growth of IoT devices has led to an unprecedented influx of data at the network edge, creating a pressing need for efficient processing and analysis capabilities. Edge computing, by decentralizing computational resources, enables data processing to occur closer to the point of data generation, thereby reducing the latency associated with transmitting data to centralized cloud servers. This paradigm shift is particularly crucial for latency-sensitive applications such as autonomous vehicles, industrial automation, and remote healthcare monitoring, where timely decision-making is paramount.

Furthermore, the inherent heterogeneity and dynamism of Edge environments, cha-

racterized by diverse hardware configurations, varying network conditions, and intermittent connectivity, pose significant challenges for traditional application deployment models. Containerization, which encapsulates applications and their dependencies into lightweight and portable containers, offers a compelling solution to these challenges. By providing a standardized environment for application execution, containers enable developers to package, deploy, and manage applications consistently across diverse Edge nodes, irrespective of underlying hardware or operating systems.

Containerization has gained widespread adoption in recent years, fueled by its ability to facilitate the development, deployment, and operation of software applications across different computing environments. At the heart of containerization is the concept of encapsulating an application and its dependencies into a single, self-contained unit called a container. Unlike traditional virtual machines, containers share the host operating system's kernel and only encapsulate the application's runtime environment and dependencies, resulting in significantly reduced overhead and improved efficiency.

In the context of Edge computing, where resources are often constrained, and reliability is paramount, containerization offers several advantages. Containers are lightweight, portable, and scalable, making them well-suited for Edge environments with limited computational resources and intermittent connectivity. Additionally, containers provide isolation between applications, ensuring that changes or failures in one container do not affect others, thereby enhancing overall system reliability and robustness. Furthermore, through the adoption of container management methodologies, particularly Kubernetes which can be regarded as a best-practice solution, there is facilitation in the effective administration of microservices reliant on containerization (referred to as pods within the context of Kubernetes terminology). (Campanile et al., 2020)

Container migration, the process of transferring a containerized application from one computing environment to another, plays a crucial role in the dynamic and heterogeneous landscape of Edge computing. In Edge environments, where nodes may join or leave the network dynamically, container migration enables load balancing, fault tolerance, and resource optimization. However, determining optimal migration decisions requires a holistic assessment of multiple interdependent factors, including hardware-level metrics (e.g., CPU and memory utilization, power consumption), network-level parameters (e.g., bandwidth availability, latency, congestion) (Abdullah et al., 2022), container-specific statistics (e.g., CPU/memory consumption, error rates), and IoT

service-level attributes (e.g., service priority, availability, and real-time performance) (Kim and Kim, 2023). Traditional rule-based or heuristic migration techniques (Tang, Luo, Obaidat and Vijayakumar, 2023) often struggle to adapt to the inherent unpredictability of IoT environments. To overcome these limitations, intelligent decision-making models leveraging deep learning (Chen et al., 2023) or fuzzy logic—such as Mamdani and Sugeno inference systems—can be developed as promising solutions. Fuzzy logic enables adaptive, context-aware decision-making by incorporating multiple uncertain and imprecise variables, allowing for more flexible and intelligent migration strategies (Campanile et al., 2020).

The primary objective of this study is to investigate the dynamics of container migration in Edge computing environments with a focus on IoT integration and Fuzzy-logic based decision making. Specifically, the study aims to explore existing approaches and techniques for container migration in Edge computing, evaluate the performance and effectiveness of container migration strategies in IoT-driven Edge environments, identify challenges and limitations associated with container migration in Edge computing with IoT context, and propose novel solutions and best practices to enhance the efficiency and reliability of container migration in such environments.

The remainder of the study is structured as follows : Chapter 2 provides a comprehensive review of related work in the field of container migration and Edge computing, highlighting existing research contributions and gaps. Chapter 3 presents the approach and the methodology employed in this study. Chapter 4 introduces the experimental setup, metrics for evaluation, and data collection procedures in IoT-driven Edge computing environment simulation. Finally, Chapter 5 discusses the findings obtained from the experimental evaluation of container migration strategies, then concludes the study by summarizing key findings, highlighting contributions, and outlining avenues for further exploration.

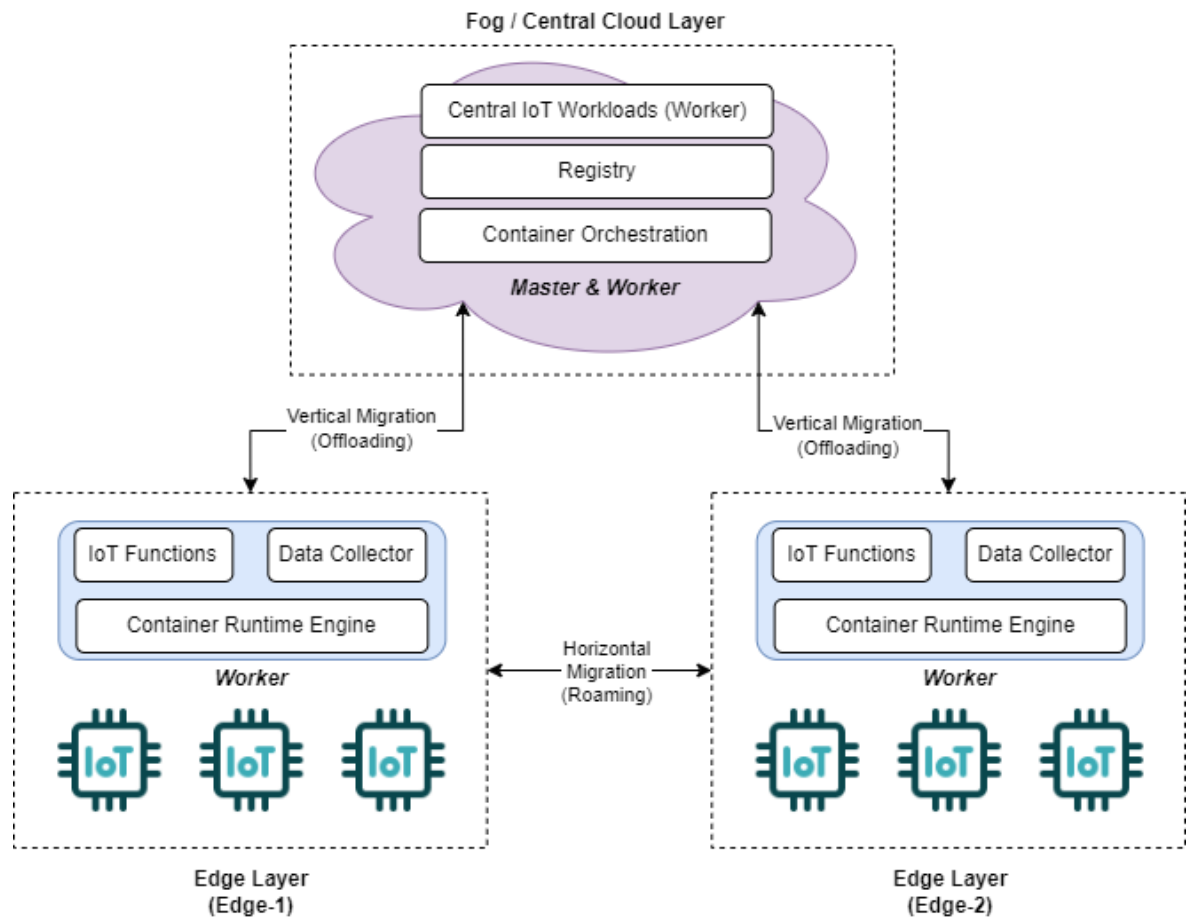


FIGURE 1.1 – Illustration of Container Workload Migration in Edge based IoT Environment

## 2 LITERATURE REVIEW

As the field of edge computing continues to grow, the management of containerized applications in IoT environments brings both challenges and innovative solutions. Edge computing's expansion supports diverse applications requiring low latency and high reliability, such as augmented reality and real-time video streaming. Researchers have developed various techniques to optimize container migration, enhance resource management, and improve overall service quality. These efforts aim to address critical issues like reducing latency, minimizing downtime, and ensuring efficient data handling.

Advancements in AI-driven scheduling algorithms and innovative orchestration methods have significantly improved the performance and reliability of edge computing systems. Additionally, the adaptation of container orchestration frameworks for edge environments has facilitated better resource utilization and service delivery. This chapter delves into these developments, offering a detailed examination of state-of-the-art solutions for managing containerized applications in edge computing, particularly within heterogeneous IoT architectures.

### 2.1 Container Migration Techniques

Container migration techniques play a crucial role in optimizing the performance and efficiency of edge computing environments. Recent studies have introduced innovative methods to address the specific challenges posed by migrating containerized applications in fog/edge computing settings. One approach, referred to as MiGrror, employs an event-driven paradigm to synchronize memory updates between source and destination, thereby reducing latency, downtime, and data loss. Another system, called H-Container, is designed to enhance the flexibility and effectiveness of container migration across heterogeneous instruction set architectures (ISA), ensuring compatibility and minimal overhead. Additionally, researchers have proposed the Container Migration-Based Decision-Making (CMDM) mechanism, which maximizes resource usage and minimizes migration impact through load balancing and migration cost considerations. A proactive approach to microservice migration in mobile edge computing has also been

introduced, leveraging reinforcement learning algorithms to improve quality of service (QoS) for various applications. Furthermore, a novel approach for online container migration in edge-assisted vehicular networks has been presented, effectively reducing total latency compared to baseline approaches. These studies collectively contribute to advancing container migration techniques tailored for edge computing environments, addressing critical requirements such as reduced latency, improved reliability, and enhanced resource utilization.

Abdullah et al. provide a novel method that addresses the need for effective container migration to improve resource management and lower latency for advanced applications. This algorithm is intended to control container execution in edge-cloud systems. The method employs live migration approaches to provide minimal disturbance and service stability during container movement across edge nodes, taking into account constraints like resource availability, deadline time, and node locations. The proposed algorithm shows efficacy in offering preliminary solution and enabling dynamic resource management, despite inherent constraints like the difficulty of locating the closest suitable node and guaranteeing data confidentiality during migration. The study emphasizes the significance of real-time restrictions and optimal response time in edge computing environments, highlighting the need to protect data by preventing unwanted access to mobile containers (Abdullah et al., 2022).

Das et al. underline the critical need for effective live migration solutions in edge clusters with limited resources. The study presents LIMOCe, a unique technique that is intended to effortlessly transfer containers between edge devices in order to address issues caused by network partitions and device failures. The study introduces DynYCSB, an adaptive benchmark suite with dynamic workload generation capabilities for edge application evaluation. By means of simulations and benchmark trials, LIMOCe exhibits better performance than current solutions, highlighting its usefulness in practical edge computing settings (Das and Sidhanta, 2023).

Rezazadeh et al. explain the novel MiGrror method for enhancing the effectiveness of virtual machine/container migration in fog/edge computing environments. With contemporary applications like augmented reality and real-time video streaming, there is a need for reduced latency, downtime, and data loss. This approach aims to meet these requirements. With MiGrror's event-driven paradigm, any memory update at the source initiates synchronization with the destination. This approach is demonstrated

by a comparison with pre-copy migration in the study, which also highlights a significant methodological difference. In order to minimize the quantity of data transported during downtime and enable more frequent updates, MiGrror dynamically synchronizes events with memory changes. According to simulation studies, MiGrror delivers the lowest average delay, minimum packet loss, and shortest downtime, making it especially beneficial for latency-sensitive applications in MEC environments. The study also emphasizes how MiGrror’s architecture can be extended to support stateless migration and integration with existing non-ML and machine learning (ML) techniques, emphasizing the platform’s adaptability to various use cases and migration situations (Rezazadeh et al., 2022).

Xing et al. introduce H-Container, a system designed to solve the challenge of migrating containerized applications in containerized IoT architectures across Heterogeneous Instruction Set Architectures (ISA). H-Container enhances the flexibility and effectiveness of server application migration within edge computing environments by utilizing LLVM-based decompiler-compiler technology and expanding CRIU-based user-space checkpoint/restore mechanisms to facilitate smoother application migration across various ISAs. H-Container’s architecture aims to ensure compatibility, minimal overhead, and ease of deployment. This enables horizontal and vertical movement between edge nodes, maximizing latency-sensitive services. The study illustrates the use of H-Container in real-world applications by performing in-depth tests on a variety of edge node configurations, such as x86-64 and arm64 servers, embedded/development boards and AWS EC2 instances. The results show notable performance improvements, with latency-sensitive applications like Redis showing a staggering 94% increase in throughput (Xing et al., 2022).

Ma et al address the load balancing issue in edge computing based on container infrastructure, especially in the context of the IoT. The research proposes a Container Migration-Based Decision-Making (CMDM) mechanism to maximize resource usage and reduce container migration impact in light of the disparate distribution of business requests among edge nodes. The three main parts of the CMDM are the migration strategy, the load balancing joint migration cost (LBJC) container migration model, and load balancing detection. The LBJC model balances migration impact by taking load balancing and migration cost into account, whereas load diversity across edge nodes is measured for load balancing detection. Using an enhanced Ant Colony System (ACS) algorithm, the migration strategy chooses containers for migration based on

resource correlation and business relevance. Simulation results demonstrate the effectiveness of CMDM, enhancing load balancing joint migration cost by 7.3% to 12.5% compared to existing approaches (Ma et al., 2020). CMDM mechanism maximizes resource usage and minimizes the impact of migration, ensuring efficient load balancing in edge computing scenarios.

Ray et al. migration of microservices in the framework of mobile edge computing (MEC). The study offers a proactive approach to handle the challenges faced by dynamic user mobility and service invocation patterns, based on reinforcement learning (RL). The proposed approach shows advantages in latency over conventional reactive tactics by utilizing RL algorithms like Dyna-Q and the Directed Acyclic Graph (DAG) structure of microservice-based applications. The proactive placement scheme's efficacy is to be confirmed by real-world dataset experimentation, with the goal of showcasing its potential to improve QoS for a range of applications. In scope of future of the study several items are mentioning, including the incorporation of capacity constraints and dynamic cost functions, as well as exploring multi-hop service placement strategies to further optimize MEC environments (Ray et al., 2020).

He et al. focus on developing a container migration approach tailored for edge-cloud systems. It aims to address challenges in migrating containers between edge devices and cloud infrastructure, considering factors like network latency, resource availability, and service continuity. Key aspects include investigating migration challenges, designing an efficient mechanism, evaluating performance, analyzing cost-effectiveness, and developing strategies for workload optimization. Ultimately, the research aims to provide insights into the feasibility and benefits of container migration in edge-cloud systems to enhance resource management and deployment efficiency (He et al., 2023).

Rausch et al. present a solution to the challenges of deploying data-intensive applications on edge systems. Acknowledging the limitations of existing serverless platforms in edge systems, the study introduce Skippy, a container scheduling system that optimizes function placement on distributed and heterogeneous clusters. Leveraging Kubernetes as a base, Skippy integrates new scheduling constraints sensitive to edge system characteristics, such as proximity to data storage and compute capabilities. Through trace-driven simulations and evaluations on the edge testbed, the study demonstrate that Skippy significantly improves task placement quality compared to existing schedulers, effectively balancing operational goals like minimizing task execution time and cloud

execution cost. Additionally, Skippy’s method for fine-tuning scheduling parameters offers flexibility and adaptability to diverse edge infrastructure scenarios (Rausch et al., 2021).

Shao et al. examine the difficulties in maintaining service reliability and continuity for Internet of Vehicles (IoV) applications in edge computing environments, considering the fast moving nature of cars and the constrained reach and resource availability of dispersed edge/fog servers. The study proposes a container-based IoV edge application migration mechanism with three essential parts to overcome these issues. In order to improve determination accuracy, it first introduces migration trigger determination algorithms for cross-border migration and service deterioration migration. These algorithms make use of traffic awareness and trajectory prediction. In order to increase the stability and adaptability of migration decisions, it also presents a model for calculating migration targets that minimizes average migration time and maximizes average service time. Finally, in order to prevent local optimal migration decisions, the research proposes a migration decision algorithm based on an enhanced artificial bee colony algorithm. The proposed approach enhances the stability and adaptability of IoV application services by reducing migration times, improving average migration times, and improving average service times, as demonstrated by simulation results. By tackling important migration issues in IoV edge computing situations and offering effective migration solutions to guarantee ongoing and reliable service delivery, this advances the industry (Shao et al., 2023).

Tang et al. introduce a novel approach for online container migration in edge-assisted vehicular networks, aiming to reduce overall task latency. Addressing the need for high-quality services in such networks, the study comprehensively models the multi-user layer-aware container migration problem, considering initialization, computation, and migration latency. A feature extraction method utilizing attention and long short-term memory (LSTM) is proposed to extract complex layer-sharing information among multiple users. Subsequently, a policy gradient-based reinforcement learning algorithm is employed for online migration decisions. Real-world data traces are used in experiments, demonstrating that the proposed algorithm effectively reduces total latency by up to 30% on average compared to baselines. The paper concludes with plans for future work, which will focus on joint optimization of layer-based container placement and migration within the Kubernetes system (Tang, Mou, Lou, Jia, Wu and Zhao, 2023).

In conclusion, the diverse range of container migration techniques discussed in this section underscores the importance of optimizing resource management and deployment efficiency in edge computing environments. These techniques address various challenges such as reducing latency, minimizing downtime, enhancing reliability, and maximizing resource utilization. By leveraging innovative approaches such as event-driven synchronization, heterogeneous ISA compatibility, load balancing mechanisms, reinforcement learning algorithms, and online migration strategies, researchers are advancing the state-of-the-art in container migration for edge computing. These advancements hold promise for enabling edge applications to meet the demanding requirements of latency-sensitive and mission-critical use cases across diverse industries. As edge computing continues to evolve, further research and development in container migration techniques will be essential to unlocking the full potential of edge computing technologies. Table 2.1 presents notable existing studies on container migration techniques.

TABLE 2.1 – Overview of Existing Studies on Container Migration Techniques

| Paper                                    | Key Contributions, Methodology, and Outcomes                                                                                                                                                                                |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (Abdullah et al., 2022)                  | Novel method for effective container migration in edge-cloud systems utilizing live migration approaches to improve resource management and lower latency. Emphasizes real-time restrictions and optimal response time.     |
| (Das and Sidhanta, 2023)                 | Introduces LIMOCe for container migration in edge clusters, addressing network partitions and device failures. Demonstrates better performance than current solutions in practical edge computing settings.                 |
| (Rezazadeh et al., 2022)                 | Presents MiGrror method for virtual machine/container migration in fog/edge computing, reducing latency, downtime, and data loss. Offers dynamic synchronization of events and memory changes for efficient migration.      |
| (Xing et al., 2022)                      | Introduces H-Container for migrating containerized applications across Heterogeneous Instruction Set Architectures (ISA) in edge computing. Shows notable performance improvements in real-world applications.              |
| (Ma et al., 2020)                        | Proposes CMDM mechanism for load balancing in edge computing based on container infrastructure, enhancing resource usage and minimizing migration impact.                                                                   |
| (Ray et al., 2020)                       | Presents a proactive approach for microservice migration in mobile edge computing (MEC), improving latency over conventional tactics using reinforcement learning algorithms.                                               |
| (He et al., 2023)                        | Develops a container migration approach for edge-cloud systems, addressing challenges in migrating containers between edge devices and cloud infrastructure. Aims to enhance resource management and deployment efficiency. |
| (Rausch et al., 2021)                    | Introduces Skippy for function placement optimization on distributed and heterogeneous clusters in edge systems. Improves task placement quality compared to existing schedulers.                                           |
| (Shao et al., 2023)                      | Examines difficulties in maintaining service reliability for IoV applications in edge computing. Proposes a container-based migration mechanism to enhance service stability and adaptability.                              |
| (Tang, Mou, Lou, Jia, Wu and Zhao, 2023) | Presents a novel approach for online container migration in edge-assisted vehicular networks to reduce task latency, employing reinforcement learning algorithms.                                                           |

## 2.2 Task Scheduling and Orchestration

In the rapidly evolving landscape of IoT and edge computing, efficient task scheduling and orchestration play pivotal roles in optimizing resource utilization, minimizing latency, and enhancing overall system performance. As these technologies become increasingly intertwined with diverse application scenarios, from industrial automation to smart cities, the need for sophisticated scheduling algorithms and orchestration mechanisms becomes more pronounced. This section delves into various studies addressing these challenges, presenting innovative approaches and methodologies aimed at streamlining task allocation, container migration, and resource management in edge and IoT environments. Through a comprehensive exploration of these contributions, we gain insights into the complexities of orchestrating tasks in distributed computing architectures, along with the potential solutions that promise to shape the future of edge and IoT systems.

Tang et al. provide a analysis of task scheduling using containers in a cutting-edge cloud-based collaboration environment for AI-driven IoT applications. The study provides a unified system concept that enables the development of IoT application services in containers by managing cloud and edge resources on a single platform. In order to choose the most efficient container for task execution, the suggested task scheduling algorithm, PGT, utilizes TOPSIS in conjunction with a priority-aware greedy strategy. It takes into account a number of factors, including task response time, energy usage, and execution cost. Experimentation involves simulations across various scenarios, influencing the number of edge servers and tasks. The outcomes demonstrate the higher performance of PGT over the four baseline algorithms, with improvements in total violation time, energy usage, penalty cost, and QoS satisfaction rate. The algorithm's effectiveness in addressing edge servers' resource limitations and reducing the high expenses linked to cloud services is further supported by the thorough analysis (Tang, Luo, Obaidat and Vijayakumar, 2023).

Mutichiro et al. present a study on how to best allocate resources and schedule tasks in containerized IoT systems, with an emphasis on edge computing settings. The difficulties of scheduling heterogeneous workloads with various resource needs and QoS restrictions are covered in research. With an emphasis on improving service time, the suggested technique, called STaSA, proposes to increase cluster resource utilization and QoS in edge cloud clusters. The methodology minimizes QoS violations by utili-

zing containerization and real-time constraints, and it outperforms previous methods in terms of performance. This research combines theoretical modeling with real-world application on the KubeEdge orchestration platform, showing improvements in system efficiency and a decrease in task queue congestion (Mutichiro et al., 2021).

Lou et al. address the challenge of minimizing container startup latency in edge computing environments. Research introduces the Container Assignment and Layer Sequencing (CALS) problem, proving its NP-hardness, and proposes the Layer-Aware Scheduling Algorithm (LASA) to optimize container assignment and layer sequencing. Additionally, LASA determines the layer download sequence on each edge node using a greedy layer sequencing algorithm. The system model encompasses multiple user equipments (UEs), heterogeneous edge nodes, a scheduler, and a container registry. Through extensive simulations and experiments, the paper demonstrates LASA’s effectiveness, achieving a substantial reduction in startup latency by 40% to 60% (Lou et al., 2022).

Singh et al. investigate the technical aspects of container migration in the context of IoT architectures, including a thorough examination of various migration strategies and their respective performance. The study uses a systematic taxonomy to classify migration methods, thereby explaining the operational differences between live and static approaches. In-depth comparisons are made, with a focus on factors such as migration time, resource utilization, and downtime. Furthermore, the evaluation includes a diverse range of methods, from cutting-edge prediction-based schemes to traditional pre-copy algorithms. The underlying architectural requirements for a successful migration are also discussed, as well as the role of network virtualization, namespaces, and cgroup subsystems in ensuring smooth host transitions. Furthermore, the paper evaluates cutting-edge technologies, offering insights into techniques such as adaptive bandwidth and CRIU-based memory migration (Singh and Singh, 2021).

Cloud data centers (DCs) are facing challenges with managing the increased load from IoT-based infrastructure, which is causing delays in fulfilling the higher data rates and lower latency requirements of IoT devices. Edge computing has become a vital solution to this problem, balancing service provisioning. The IoT-edge-cloud ecosystem’s interactions and data transmissions across layers, however, may put a strain on network infrastructure. Although there is opportunity in software-defined edge computing, current methods have difficulty achieving a compromise between latency and energy economy. (Singh et al., 2021) propose a software-defined edge computing (SDE)-

based, energy-efficient, and lightweight container-as-a-service (CaaS) solution to manage latency-sensitive IoT workloads. It develops a two-period resource allocation Stackelberg game between edge devices and end users/IoT devices while taking service level agreements into account. The study additionally establishes an energy-efficient group for load balancing in the software-defined edge computing environment through container allocation, consolidation, and migration. The outcomes of the simulation show that the suggested method outperforms the current variations in terms of CPU serve time, network serve time, total delay, and energy usage.

Marchese et al. propose enhancements to Kubernetes for optimizing application migration in containerized IoT architectures, focusing on Fog and Edge Computing environments. Research introduces a network-aware scheduler plugin and a custom descheduler to address challenges related to variable network conditions and communication requirements. The scheduler plugin extends Kubernetes' default scheduler to consider node-to-node latencies and microservice communication during Pod placement, while the descheduler monitors network conditions and migrates Pods for better placement at runtime. Experimental evaluation using the Bookinfo sample application demonstrates significant reductions in end-to-end response time compared to the default Kubernetes scheduler. Future research directions include incorporating AI techniques, such as Reinforcement Learning, to further improve scheduling and descheduling algorithms based on historical network and application traffic metrics (Marchese and Tomarchio, 2022).

Fard et al. examine the difficulties and solutions related to the provisioning of resources in containerized edge & IoT architectures. The study emphasizes how typical centralized cloud architectures are unable to handle the enormous volume of IoT data, which results in higher latency and computational bottlenecks. An edge-fog-cloud approach that makes use of containerization technologies for effective resource management has been presented as a solution to these issues. The method involves applying an application-aware container scheduler that takes into account system dynamics, computing capability, and proximity to data producers and consumers in order to orchestrate dynamic heterogeneous resources across edge and fog systems. The research shows through a thorough examination how well their suggested container orchestration mechanism works to minimize IoT service response times, which enhances overall system performance and lowers use costs. Moreover, comparison of the proposed approach with existing solutions has been conducted, highlighting its superiority in terms of responsiveness and resource utilization (Fard et al., 2020).

Oleghe et al. investigate the migration and placement of containers in edge computing in order to fulfill the growing need for fast, low-latency compute. In managing container-based applications on heterogeneous computing nodes, the study highlights the importance of container schedulers and their role in resource allocation and migration decision optimization. The study emphasizes the frameworks and algorithms utilized in scheduling models, providing insight into a number of strategies, such as graph-based techniques, heuristic algorithms, and optimization modeling. The study highlights the complicated nature of the container placement issue, which is frequently simplified by the use of graph network models or multi-objective optimization in order to enable computational solutions. Despite the fact that heuristic-based algorithms still dominate most scheduling techniques, the study indicates that interest in decentralized scheduling systems—especially for mobile edge nodes—is growing. However, the study acknowledges a scarcity of models considering distributed edge computing tasks, highlighting the need for further research in this area (Oleghe, 2021).

Aleydeh et al. demonstrate how container orchestration has been optimized in a hybrid computing setting to address the growing need for low-latency services. It suggests the Edge Computing-enabled Container Migration/Re-Installation (EC2-MRI) model and the Optimal Container Migration/Re-Installation (OC-MRI) model to address adoption challenges for edge computing. While the EC2-MRI model offers a heuristic-based solution for real-time implementation, the OC-MRI model uses an integer programming optimization method to decrease downtime and latency. Both models show reduced downtime and decreased latency when compared to traditional methods; the OC-MRI model has the lowest latencies. The study highlights the necessity for scalable and effective container orchestration techniques by emphasizing the crucial role that appropriate edge service placement plays for network service providers and end users. Future research aims to enhance the models' scalability and real-time capabilities, potentially incorporating multi-objective optimization and machine learning-based approaches (Aleyadeh et al., 2022).

Barbalace et al. present H-Container, a novel method that makes it easier to migrate natively-compiled containerized applications between compute nodes with CPUs that have varied Instruction Set Architectures (ISAs) in edge computing environments. H-Container provides highly effective compatibility, deployability, and Linux compliance without requiring source code or compiler adjustments, thereby addressing the problem of heterogeneous ISA nodes preventing server relocation for preserving client-

server proximity. By converting executable binaries and utilizing a user-space checkpoint/restart framework, the study leverages LLVM and CRIU technologies to facilitate a smooth migration experience. Experimental results show that there is little cost during program execution and migration, and that there are real benefits—a 94% boost in Redis throughput, for example. This novel approach demonstrates the possibility of achieving improved performance in applications that are sensitive to latency in heterogeneous edge computing settings (Barbalace et al., 2020).

In conclusion, the diverse array of studies discussed in this section underscores the critical role of task scheduling and orchestration in containerized edge computing environments. These studies have introduced innovative algorithms, techniques, and frameworks to address various challenges such as latency reduction, resource allocation optimization, and workload management. By leveraging advanced scheduling algorithms, such as Priority-Greedy Task (PGT) and Layer-Aware Scheduling Algorithm (LASA), researchers have demonstrated significant improvements in system efficiency, reduced latency, and enhanced resource utilization. Moreover, the exploration of container migration strategies and software-defined edge computing solutions has provided valuable insights into addressing complexities associated with data transmission and network infrastructure strain. Overall, these studies contribute to advancing the state-of-the-art in task scheduling and orchestration for containerized edge computing, paving the way for more efficient, scalable, and reliable edge computing systems in the future. Table 2.2 presents notable existing studies on task scheduling and orchestration.

TABLE 2.2 – Overview of Existing Studies on Task Scheduling and Orchestration

| Paper                                      | Key Contributions, Methodology, and Outcomes                                                                                                                                                                                                                     |
|--------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (Tang, Luo, Obaidat and Vijayakumar, 2023) | Unified system concept enabling IoT application services development in containers. Proposed task scheduling algorithm PGT utilizes TOPSIS with a priority-aware greedy strategy, demonstrating higher performance over baseline algorithms.                     |
| (Mutichiro et al., 2021)                   | STaSA technique for resource allocation and task scheduling in containerized IoT systems, improving service time and cluster resource utilization. Combines theoretical modeling with real-world application on KubeEdge platform.                               |
| (Lou et al., 2022)                         | Introduces CALS problem and LASA algorithm to minimize container startup latency in edge computing environments. Achieves significant reduction in startup latency through greedy layer sequencing algorithm.                                                    |
| (Singh and Singh, 2021)                    | Systematic taxonomy of container migration strategies in IoT architectures, evaluating operational differences and architectural requirements. Discusses advancements in migration technologies for smooth host transitions.                                     |
| (Singh et al., 2021)                       | Proposes SDE-based CaaS solution for managing latency-sensitive IoT workloads in edge computing. Utilizes two-period resource allocation Stackelberg game for energy-efficient load balancing.                                                                   |
| (Marchese and Tomarchio, 2022)             | Enhancements to Kubernetes for optimizing application migration in containerized IoT architectures. Introduces network-aware scheduler plugin and custom descheduler for better placement at runtime.                                                            |
| (Fard et al., 2020)                        | Examines resource provisioning challenges in containerized edge & IoT architectures. Proposes application-aware container scheduler for efficient resource management, improving system performance and reducing costs.                                          |
| (Oleghe, 2021)                             | Investigates container migration and placement in edge computing, emphasizing the role of container schedulers and their optimization. Highlights the need for further research in distributed edge computing tasks.                                             |
| (Aleyadeh et al., 2022)                    | Introduces EC2-MRI and OC-MRI models for optimizing container orchestration in hybrid computing. Shows reduced downtime and latency compared to traditional methods, highlighting the importance of appropriate edge service placement.                          |
| (Barbalace et al., 2020)                   | Presents H-Container for migrating natively-compiled containerized applications between compute nodes with varied ISAs in edge computing environments. Achieves improved performance in latency-sensitive applications in heterogeneous edge computing settings. |

### 2.3 Edge Computing Frameworks and Architectures

In the rapidly evolving landscape of edge computing, the efficient management of complex and heterogeneous computing environments has emerged as a pivotal challenge for service and infrastructure providers. As the convergence of cloud and IoT technologies continues to reshape modern computing paradigms, advanced orchestration systems play a crucial role in supporting resource management across large-scale data centers integrated with the burgeoning data streams generated by IoT devices. These systems must possess the capability to dynamically allocate resources and deploy applications in real-time, leveraging the available computational resources across cloud, edge, and fog layers. This section explores a diverse array of studies that delve into the intricacies of edge computing frameworks and architectures, ranging from container-based orchestration solutions to innovative approaches for optimizing service deployment and migration. Through detailed investigations and experimental evaluations, these studies shed light on key challenges and propose novel solutions aimed at enhancing scalability, fault tolerance, and performance in edge computing environments.

As edge computing gains momentum, IT infrastructure becomes more decentralized, supporting server-class compute nodes in addition to devices with limited resources. These nodes, which are capable of running workloads that are comparable to those in the cloud, are positioned in regional data centers or close to wireless base stations. However the orchestration tools for cloud containers that are now in place have not yet adjusted to this new edge cloud environment. By presenting EdgeNet, a Kubernetes-based edge cloud testbed based on containers, (Şenel, 2023) offer an approach. The study discusses the necessity of multitenancy in edge environments with restricted resources and presents empirical data supporting the use of a single-instance multitenancy framework. Furthermore, an in-cluster VPN for nodes behind NATs and workload deployment depending on node location are two more features and approaches that the study offers to customize container orchestration for the edge cloud. EdgeNet serves as a globally distributed compute cluster, offering a cost-effective and easier-to-maintain alternative to previous testbeds, thereby advancing container orchestration capabilities in the edge cloud.

The effective management of complex and heterogeneous computing environments is a significant challenge for service and infrastructure providers in the cloud-to-thing continuum era. Advanced orchestration systems are crucial to support resource management

in large-scale cloud data centers integrated with the big data generation of IoT devices. These systems should be aware of all available resources and their current status to enable dynamic allocations and short-time deployment of applications. (Svorobej et al., 2020) review the state of the art in orchestration along the cloud-to-thing continuum, with a specific emphasis on container-based orchestration (e.g., Docker Swarm and Kubernetes) and fog-specific orchestration architectures.

Liu et al. examine how well edge-cloud computing infrastructures based on containers perform for industrial Internet of things (IIoT) applications. It uses an edge-cloud computing architecture based on containerization, whereby industry applications are divided into separate containers and runtimes are handled at the edge unit. Container orchestration solutions such as Kubernetes enable these containerized apps to be created, upgraded, and terminated remotely in an elastic way. The architecture also comprises an edge hub infrastructure that is in charge of routing messages among containerized apps, enabling bidirectional communication between field devices and the cloud platform, and managing traffic flows inside the edge. The study emphasizes the promising potential of containerization on the edge for edge-cloud computing through performance evaluation, including measurements of entire stack latency and processing capability for machine learning workloads. It does, however, point out that there is still much space for improvement when it comes to satisfying the performance requirements of time-sensitive industrial applications, highlighting the need to optimize industrial apps by dividing them into microservices throughout the edge-cloud stack. The study's evaluation approach offers developers and users in the edge-cloud computing space insightful information (Liu et al., 2021).

Scolati et al. investigate data stream processing containerized edge cloud architecture, putting a lot of focus on experimental findings and technical specifics. The system is decomposed into layers ; Apache Hadoop-Spark handles data, Docker containers handle software orchestration and deployment, and Raspberry Pis are used as the device layer. The study demonstrates the improved fault tolerance and easier maintenance associated with containerization through extensive experimental evaluations. Furthermore, a thorough assessment of the architecture's ability to handle resource limitations is carried out, with a specific emphasis on CPU and memory usage. In order to offer solutions, the study focuses on the difficulties associated with utilizing the Hadoop Distributed File System (HDFS) in environments with limited resources. Additionally, trust management solutions utilizing blockchain technologies are explored, highligh-

ting their role in ensuring secure interactions across organizational boundaries (Scolati et al., 2020).

Aguzzi et al. explore the extension of the Web of Things (WoT) standard to accommodate the dynamic nature of IoT environments within cloud-edge continuum deployments. While the WoT standard provides a framework for interoperable IoT systems, it lacks provisions for dynamic allocation and migration of Web Things (WTs) across computational nodes. The proposed Migratable WoT (M-WoT) framework introduces live migration capabilities for WTs, leveraging well-defined WT interfaces to facilitate dynamic allocation based on interaction patterns. The study presents a novel architecture for M-WoT, formulates WT allocation as a multi-objective optimization problem, and implements a container-based solution. Evaluation results demonstrate the effectiveness of the proposed migration policy in distributed edge computing setups and real-world IoT monitoring scenarios, paving the way for future extensions focusing on scalability, fault tolerance, and adaptive deployment strategies using machine learning techniques (Aguzzi et al., 2020).

Alam et al. propose a modular and scalable architecture based on lightweight virtualization to address the challenges of developing architectures for Internet of Things (IoT) applications. Emphasizing the need for timeliness, low latency, and high reliability in future IoT architectures, the proposed architecture leverages edge computing to move services directly to end devices and cache information locally. The architecture consists of three main layers : sensing, mediation, and enterprise layers, spanning across cloud, fog, and edge devices. Services are containerized and distributed across these layers, ensuring efficient data collection, analysis, and computation. Orchestration rules enable full service recovery during failures, while redundancy at different layers enhances system resilience. Experimental results validate the architecture’s effectiveness in deploying on-demand services across different layers and improving application latency by migrating services to the edge layer. The study showcases the architecture’s ability to ensure higher reliability, availability, and performance in IoT deployments, demonstrated through a smart home use case (Alam et al., 2018).

Bulej et al. present a method for soft real-time guarantees on the microservices’ response time that are deployed in edge-cloud environments. This method facilitates the process without requiring specific programming models or explicit reservations by allowing developers to define desired response time assurances in probabilistic terms.

The study addresses the need for real-time responsiveness in current Cyber-Physical Systems (CPS). The strategy minimizes developer impact while ensuring microservices achieve performance requirements. It does this by using an adaptive control loop for deployment management and statistical methods to forecast response time uppers. Notably, the method does not force developers to use particular software architectures or programming languages; instead, it works within non-overcommitted system states to preserve accuracy in reaction time forecasts. Future plans include incorporating network topology and latencies into the control loop for more flexible microservice distribution across edge-cloud nodes (Bulej et al., 2021).

Chen et al. investigate the dynamic task allocation and service migration (DTASM) problem in edge-cloud IoT systems, where computing requirements and user mobility change over time. It addresses the challenge of maintaining seamless service migration while minimizing cloud server load and ensuring low latency. The problem is formulated as a Markov Decision Process (MDP), and a deep reinforcement learning (DRL) approach based on twin-delayed deep deterministic policy gradient (DDPG) is proposed to provide a solution to the vast discrete action space challenge. The DTASM problem is decomposed into user selection and task allocation sub-problems, and the proposed approach optimizes the task allocation policy to minimize load forwarded to the cloud while satisfying seamless migration, service latency, and computing capacity constraints. Extensive simulations demonstrate that the proposed DRL-based algorithm is a better option in maintaining stable QoS and exceeds the performance thresholds of other benchmarks in terms of load forwarding, service downtime, migrated users, and migration-failed users. The study contributes to addressing DTASM in large-scale IoT systems with dynamic requirements and mobility, providing insights into seamless service migration and dynamic task allocation strategies (Chen and Zhou, 2021).

Besides, some researchers present a novel solution to the complicated challenge to provide solution in scope of dynamic task allocation and service migration (DTASM) in edge-cloud IoT systems. It introduces the necessity of DTASM in ensuring QoS and resource utilization in environments with heterogeneous resource capacities and dynamic computing requirements. The paper proposes a deep reinforcement learning (DRL) approach, formulated as a Markov Decision Process (MDP), to address the DTASM problem. By decomposing the problem into user selection and task allocation sub-problems and employing a twin-delayed deep deterministic policy gradient (DDPG)

architecture, the proposed approach effectively tackles the challenge of vast discrete action spaces. Through extensive simulations, the paper demonstrates the superiority of the DRL-based approach in maintaining QoS and minimizing overheads compared to benchmarks. Research provides a comprehensive consideration of constraints such as seamless migration and service latency and further validation in real-world environments and exploration of scalability aspects would enhance its applicability and impact (Chen et al., 2022).

Also, they emphasize cost-effective resource optimization, real-time service processing, and eliminating edge server waste. The study dramatically reduces resource consumption and deployment cost over current approaches by putting forth a simple and fault-tolerant deployment model enhanced by an Improved Genetic-Simulated Annealing Algorithm (IGSAA). It emphasizes how edge computing can improve service quality and decrease latency, particularly in settings like intelligent factories. The research offers new insights into improving edge computing performance and reliability by taking into account numerous optimization objectives and resolving hardware-induced deployment issues (Chen et al., 2023).

Dupont et al. present a comprehensive exploration of containerized IoT functions/tasks migration and orchestration within hybrid cloud-edge architectures. In response to the escalating demands for low-latency services and the challenges posed by the proliferation of IoT devices, associated data streams & increasing complexity of heterogeneous IoT environments, the authors introduce the Cloud4IoT platform. This platform facilitates the deployment and migration of containerized functionalities across IoT gateways, edge, and cloud layers. Notably, Cloud4IoT enables both horizontal migration (roaming) between gateways and vertical migration (offloading) between the edge and cloud layers. Leveraging containerization (Linux containers), Cloud4IoT achieves automatic deployment, dynamic configuration, and orchestration of IoT functions, thereby enhancing scalability and efficiency. The study has been implemented via utilization of various technologies for experiment including the use of OpenStack and Kubernetes, as well as the deployment on IoT gateways such as Raspberry Pi. Through two illustrative use cases - IoT roaming in healthcare and IoT offloading for remote engine diagnostics – research demonstrates the practical applications and benefits of Cloud4IoT. In conclusion, the study highlights the potential of Fog computing solutions in addressing current IoT challenges and outline future research directions, including the exploration of Infrastructure as Code (IaaS) deployment techniques and security considerations for

containerized IoT functions (Dupont et al., 2017).

Kim et al. provide a technical evaluation of KubeEdge’s effectiveness in containerized edge computing environments, focusing on problems with edge node delays and the distribution of computational resources. Experiments with KubeEdge clusters with cloud and edge nodes were carried out, and the results showed that traffic forwarding between far-off edge nodes significantly reduced cluster throughput. They suggest a local scheduling strategy in which user traffic is handled locally at each edge node rather than being routed remotely in order to lessen this. This approach reduces latency and increases throughput, showing that it can be used to address performance issues related to traffic forwarding and geographic distribution in edge computing setups. Furthermore, the study assesses KubeEdge’s orchestration performance for containerized IoT application services, identifies cloud-based system restrictions, and suggests a local scheduling approach to minimize service delay and maximize resource utilization. The suggested scheme’s efficacy in outperforming the round-robin scheme in terms of cumulative throughput and response time over a range of traffic patterns is indicated by experimental results, indicating that it may be used to optimize edge computing performance (Kim and Kim, 2023).

Nakanishi et al. propose an architecture that leverages BGP AS path information within container orchestration systems, such as Kubernetes, to optimize content delivery in edge computing environments. This architecture dynamically deploys content cache containers at edge locations based on AS path length, effectively reducing network latency and enhancing resource utilization. The study details the architecture’s components, including Kubernetes clusters distributed across edge and cloud environments, cache containers deployed at strategic locations, and a service mesh for seamless communication between containers. By integrating these components, the architecture ensures efficient content distribution while considering network topology and user proximity. Through experimental evaluation, the study demonstrates the architecture’s effectiveness in reducing response times and improving user experience (Nakanishi et al., 2020).

In conclusion, the exploration of edge computing frameworks and architectures for containerized applications reflects a growing interest in optimizing resource management and deployment efficiency in edge environments. The studies discussed in this section highlight the diverse approaches and solutions proposed to address the unique

challenges posed by edge computing, such as resource constraints, latency sensitivity, and heterogeneity. From EdgeNet's Kubernetes-based testbed to the investigation of containerized edge-cloud computing infrastructures for industrial IoT applications, researchers are actively developing and evaluating solutions to enable efficient orchestration and deployment of containerized applications at the edge. Additionally, the focus on technical specifics, fault tolerance, and trust management solutions underscores the complexity of edge computing environments and the need for comprehensive approaches to address these challenges. As edge computing continues to evolve, these studies provide valuable insights and frameworks for designing and implementing container orchestration solutions tailored to the requirements of edge computing environments. Table 2.3 presents notable existing studies on edge computing frameworks and architectures.

The literature review chapter provides a comprehensive overview of research efforts in three critical domains of edge computing : Container Migration Techniques, Task Scheduling and Orchestration, and Edge Computing Frameworks and Architectures. Across these domains, a common theme emerges : the relentless pursuit of efficiency, scalability, and performance optimization in the face of the evolving demands of modern computing environments.

In the realm of Container Migration Techniques, researchers have explored innovative methods to facilitate seamless migration of containerized applications across edge nodes, cloud environments, and IoT architectures. From live migration algorithms to event-driven synchronization mechanisms, these studies seek to minimize downtime, reduce latency, and enhance resource utilization while ensuring service stability and data confidentiality.

Task Scheduling and Orchestration constitute another crucial aspect of edge computing, where the effective allocation of computational resources and the orchestration of application tasks play a pivotal role in optimizing system performance and meeting Quality of Service (QoS) requirements. Through advanced scheduling algorithms, reinforcement learning approaches, and container-based orchestration solutions, researchers aim to dynamically adapt to changing workloads, minimize response times, and maximize resource utilization across distributed edge-cloud infrastructures.

Finally, Edge Computing Frameworks and Architectures encompass a diverse array of

TABLE 2.3 – Overview of Existing Studies on Edge Computing Frameworks and Architectures

| Paper                    | Key Contributions, Methodology, and Outcomes                                                                                                                                                                                                          |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (Svorobej et al., 2020)  | Review of orchestration systems along the cloud-to-thing continuum, focusing on container-based orchestration and fog-specific architectures.                                                                                                         |
| (Liu et al., 2021)       | Evaluation of edge-cloud computing infrastructure for industrial IoT applications, emphasizing containerization and orchestration solutions like Kubernetes.                                                                                          |
| (Scolati et al., 2020)   | Investigation of data stream processing in containerized edge cloud architecture, with a focus on fault tolerance, maintenance, and resource limitations.                                                                                             |
| (Aguzzi et al., 2020)    | Introduction of the Migratable WoT (M-WoT) framework for dynamic allocation and migration of Web Things (WTs) in IoT environments, enhancing scalability and deployment flexibility.                                                                  |
| (Alam et al., 2018)      | Proposal of a modular architecture for IoT applications leveraging edge computing, emphasizing timeliness, low latency, and high reliability, with experimental validation in smart home scenarios.                                                   |
| (Bulej et al., 2021)     | Presentation of a method for soft real-time guarantees on microservices' response time in edge-cloud environments, utilizing adaptive control loops and statistical methods for deployment management.                                                |
| (Chen and Zhou, 2021)    | Investigation of dynamic task allocation and service migration in edge-cloud IoT systems, employing deep reinforcement learning to optimize task allocation policies while ensuring low latency and QoS.                                              |
| (Chen et al., 2022)      | Introduction of a novel deep reinforcement learning approach for dynamic task allocation and service migration in large-scale IoT systems, addressing challenges of dynamic requirements and mobility.                                                |
| (Chen et al., 2023)      | Proposal of a cost-effective resource optimization model for edge-cloud IoT systems, enhancing service quality and decreasing latency through improved deployment strategies and optimization objectives.                                             |
| (Dupont et al., 2017)    | Exploration of containerized IoT functions/tasks migration and orchestration within hybrid cloud-edge architectures, introducing the Cloud4IoT platform for seamless deployment and migration across IoT layers.                                      |
| (Kim and Kim, 2023)      | Technical evaluation of KubeEdge's effectiveness in containerized edge computing environments, addressing edge node delays and proposing local scheduling strategies for performance optimization.                                                    |
| (Nakanishi et al., 2020) | Proposal of an architecture leveraging BGP AS path information for optimized content delivery in edge computing environments, demonstrating improved network latency and resource utilization through dynamic deployment of content cache containers. |

studies focused on designing scalable, resilient, and efficient architectures to support edge computing deployments. From container-based edge-cloud architectures to dynamic task allocation mechanisms, these studies strive to harness the potential of edge computing to deliver low-latency services, improve scalability, and optimize resource utilization in IoT environments.

Overall, these various approaches highlight the importance of innovative containerization and orchestration techniques in optimizing resource management and deployment efficiency within edge computing environments, especially in IoT architectures with heterogeneous ISAs. The advancements in containerized edge computing highlighted reflect a concerted effort to address the unique challenges posed by decentralized and heterogeneous computing environments. Through innovative migration techniques, optimized task scheduling, and advanced orchestration frameworks, these research efforts contribute to the development of more efficient, reliable, and scalable edge computing systems. As edge computing continues to evolve, ongoing research and development will be crucial in meeting the growing demands of IoT and other latency-sensitive applications. Table 2.4 presents the major existing studies in this domain.

In conclusion, the collective body of research presented in these three domains underscores the multifaceted nature of edge computing and the pressing need for innovative solutions to address its inherent challenges. By leveraging advances in containerization, scheduling algorithms, and architectural design, researchers are poised to unlock the full potential of edge computing, ushering in a new era of efficient, responsive, and scalable computing paradigms.

TABLE 2.4 – Overview of existing studies

| Study                                      | Key Contributions                                                                                                                                                                                                                                    | Technologies Used                                                                                                        |
|--------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| (Abdullah et al., 2022)                    | Algorithm for efficient container migration in edge-cloud systems to optimize resource management and reduce latency for advanced applications.                                                                                                      | Live migration techniques, resource availability, deadline time, node locations                                          |
| (Aguzzi et al., 2020)                      | Extension of Web of Things (WoT) standard to accommodate dynamic IoT environments within cloud-edge continuum deployments, including live migration capabilities for Web Things (WTs).                                                               | Migratable WoT (M-WoT) framework, container-based solution, multi-objective optimization                                 |
| (Aleyadeh et al., 2022)                    | Optimization of container orchestration in hybrid computing environments to meet low-latency service demands, introducing the OC-MRI and EC2-MRI models.                                                                                             | Integer programming optimization, heuristic-based solution, CPU and memory usage                                         |
| (Alam et al., 2018)                        | Modular and scalable architecture for IoT applications leveraging lightweight virtualization, focusing on time-liness, low latency, and high reliability.                                                                                            | Edge computing, lightweight virtualization, sensing, mediation, and enterprise layers                                    |
| (Barbalace et al., 2020)                   | Introduction of H-Container, facilitating migration of natively-compiled containerized applications across edge nodes with different Instruction Set Architectures (ISAs) in edge computing environments.                                            | LLVM, CRIU, user-space checkpoint/restart framework                                                                      |
| (Bulej et al., 2021)                       | Approach for providing soft real-time guarantees on microservice response time in edge-cloud environments using probabilistic terms and adaptive control loops.                                                                                      | Statistical methods, adaptive control loop, real-time responsiveness                                                     |
| (Chen and Zhou, 2021)                      | Investigation of dynamic task allocation and service migration (DTASM) problem in edge-cloud IoT systems, proposing a deep reinforcement learning (DRL) approach for optimization.                                                                   | Deep reinforcement learning (DRL), twin-delayed deep deterministic policy gradient (DDPG), Markov Decision Process (MDP) |
| (Chen et al., 2022)                        | Solution for DTASM in edge-cloud IoT systems using DRL approach, addressing resource limitations and dynamic computing requirements.                                                                                                                 | Deep reinforcement learning (DRL), twin-delayed deep deterministic policy gradient (DDPG), Markov Decision Process (MDP) |
| (Chen et al., 2023)                        | Proposes a novel solution for DTASM problem in edge-cloud IoT systems using DRL approach, emphasizing QoS, resource utilization, and dynamic computing requirements.                                                                                 | Deep reinforcement learning (DRL), twin-delayed deep deterministic policy gradient (DDPG), Markov Decision Process (MDP) |
| (Das and Sidhanta, 2023)                   | Efficient live migration solutions in resource-constrained edge clusters using LIMOCE algorithm, accompanied by Dyn-YCSB benchmark suite for edge applications.                                                                                      | LIMOCE algorithm, Dyn-YCSB benchmark suite, live migration                                                               |
| (Dupont et al., 2017)                      | Exploration of containerized IoT functions/tasks migration and orchestration within hybrid cloud-edge architectures, introducing the Cloud4IoT platform.                                                                                             | Cloud4IoT platform, containerization, IoT roaming, IoT offloading                                                        |
| (Fard et al., 2020)                        | Investigation of resource provisioning in containerized IoT architectures spanning edge and fog computing, proposing an edge-fog-cloud model for efficient resource management.                                                                      | Containerization, edge-fog-cloud model, Kubernetes, computational capacity                                               |
| (He et al., 2023)                          | Focuses on developing a container migration approach tailored for edge-cloud systems, addressing challenges in migrating containers between edge devices and cloud infrastructure.                                                                   | Container migration, resource management, deployment efficiency                                                          |
| (Kim and Kim, 2023)                        | Technical evaluation of KubeEdge's performance in containerized edge computing environments, proposing a local scheduling scheme to optimize resource utilization and reduce latency.                                                                | KubeEdge, local scheduling scheme, edge computing                                                                        |
| (Liu et al., 2021)                         | Investigation of performance of container-based edge-cloud computing infrastructures in industrial IoT (IIoT) applications, focusing on latency, reliability, and performance.                                                                       | Containerization, edge-cloud computing, Kubernetes, industrial IoT                                                       |
| (Lou et al., 2022)                         | Addresses the challenge of minimizing container startup latency in edge computing environments, proposing the CALS problem and the LASA algorithm for container assignment and layer sequencing.                                                     | Layer-Aware Scheduling Algorithm (LASA), container assignment, layer sequencing                                          |
| (Ma et al., 2020)                          | Addresses the challenge of load balancing in container infrastructure-based edge computing for IoT applications, proposing the CMDM mechanism to optimize resource utilization and minimize migration impact.                                        | Load balancing, container migration, IoT applications                                                                    |
| (Marchese and Tomarchio, 2022)             | Enhancements to Kubernetes for optimizing application migration in containerized IoT architectures, introducing a network-aware scheduler plugin and a custom descheduler.                                                                           | Kubernetes, network-aware scheduler plugin, custom descheduler                                                           |
| (Mutichiro et al., 2021)                   | Investigation into optimizing task scheduling and resource allocation in containerized IoT systems, proposing the STaSA algorithm to enhance cluster resource utilization and QoS.                                                                   | Containerization, task scheduling, resource allocation                                                                   |
| (Nakanishi et al., 2020)                   | Architecture leveraging BGP AS path information within container orchestration systems to optimize content delivery in edge computing environments.                                                                                                  | Container orchestration, BGP AS path information, content delivery                                                       |
| (Oleghe, 2021)                             | Investigation of container placement and migration in edge computing, emphasizing the role of container schedulers in optimizing resource allocation and migration decisions.                                                                        | Container placement, migration, container schedulers                                                                     |
| (Rausch et al., 2021)                      | Solution for deploying data-intensive applications on edge systems using Skippy, a container scheduling system that optimizes function placement on distributed and heterogeneous clusters.                                                          | Skippy, container scheduling, function placement                                                                         |
| (Ray et al., 2020)                         | Investigation into optimizing microservice placement and migration in Mobile Edge Computing (MEC) using proactive mechanism based on Reinforcement Learning (RL).                                                                                    | Reinforcement learning (RL), microservice placement, Mobile Edge Computing (MEC)                                         |
| (Rezazadeh et al., 2022)                   | Introduction of MiGrror method for improving efficiency of VM/container migration in fog/edge computing environments, focusing on minimizing latency, downtime, and data loss.                                                                       | MiGrror method, VM/container migration, fog/edge computing                                                               |
| (Shao et al., 2023)                        | Investigation of container-based IoV edge application migration mechanism to ensure service continuity and stability, proposing migration trigger determination algorithms and migration decision algorithms.                                        | IoV edge application migration, migration trigger determination, migration decision algorithms                           |
| (Singh et al., 2021)                       | Proposal for lightweight and energy-efficient container-as-a-service (CaaS) approach based on software-defined edge computing to handle latency-sensitive IoT workloads.                                                                             | Software-defined edge computing, container-as-a-service (CaaS), IoT workloads                                            |
| (Singh and Singh, 2021)Singh et al. (2021) | Investigation of container migration strategies in IoT architectures, providing a systematic taxonomy to classify migration methods and evaluating their performance.                                                                                | Container migration, IoT architectures, taxonomy                                                                         |
| (Scolati et al., 2020)                     | Investigation of data stream processing containerized edge cloud architecture, emphasizing fault tolerance, resource limitations, and trust management solutions.                                                                                    | Containerization, data stream processing, fault tolerance, trust management                                              |
| (Tang, Luo, Obaidat and Vijayakumar, 2023) | Introduction of a novel approach for online container migration in edge-assisted vehicular networks, focusing on reducing overall task latency using feature extraction method and policy gradient-based reinforcement learning algorithm.           | Online container migration, vehicular networks, feature extraction, reinforcement learning                               |
| (Tang, Mou, Lou, Jia, Wu and Zhao, 2023)   | Analysis of task scheduling using containers in cloud-based collaboration environment for AI-driven IoT applications, proposing PGT algorithm to optimize task execution efficiency based on response time, energy usage, and QoS satisfaction rate. | Task scheduling, AI-driven IoT applications, PGT algorithm                                                               |
| (Xing et al., 2022)                        | Introduction of H-Container, facilitating migration of containerized applications across edge nodes with different ISAs in edge computing environments, leading to performance improvements in latency-sensitive applications.                       | H-Container, LLVM, CRIU                                                                                                  |

### 3 PROPOSED APPROACH

#### 3.1 Overview

This research addresses two critical challenges in the realm of IoT systems : (1) developing an infrastructure that transcends centralized computing by enabling dynamic service management and optimization through edge-level processing in a container-based architecture, and (2) implementing efficient decision-making and orchestration mechanisms for diverse service-oriented scenarios, focusing on horizontal and vertical workload migrations for optimization, fault recovery, and mobility. These goals converge in an innovative IoT project designed to leverage containerization for live migration of workloads, ensuring seamless and adaptive service delivery.

The architecture comprises a centralized Kubernetes-based IoT cloud, which serves dual roles as a master node for orchestration and a worker node for centralized IoT workloads. The centralized cloud also hosts a service registry, embodying a service-based architecture (SBA). This registry monitors metadata related to IoT services from connected devices, enabling dynamic discovery and interaction among services while supporting real-time decision-making for workload management. At the edge, IoT gateways function as worker nodes, hosting IoT workloads as containers or pods and serving as communication hubs for IoT end-devices.

To facilitate live migration, the project employs CRIU (Checkpoint/Restore in Userspace), enabling seamless horizontal migrations (edge-to-edge) and vertical migrations (edge-to-central). A fuzzy logic-based decision-making module leverages multi-layered metrics—spanning pod/container, network, hardware, and IoT service layers—collected via exporters at both the edge and central levels. These metrics inform the dynamic determination of optimal migration actions based on predefined thresholds and decision rules.

An integrated service orchestration module executes these decisions within the containerized IoT environment, addressing scenarios such as failover of containers, pods, or nodes; resource optimization for performance and efficiency; deployment of new workloads; energy-saving strategies; and adaptation to device mobility. This modu-

lar approach ensures a holistic system capable of maintaining service continuity and optimizing resource usage under dynamic conditions.

Evaluation of the system involves a comparative analysis of various fuzzy logic algorithms, assessing their performance across key parameters such as latency, average migration time, and downtime. Additionally, the effectiveness of the proposed approach is measured against multiple migration scenarios, providing insights into its robustness and adaptability. This project not only demonstrates the feasibility of containerized IoT systems with edge and central processing but also establishes a framework for optimizing service delivery through intelligent orchestration and decision-making, catering to the evolving demands of IoT environments.

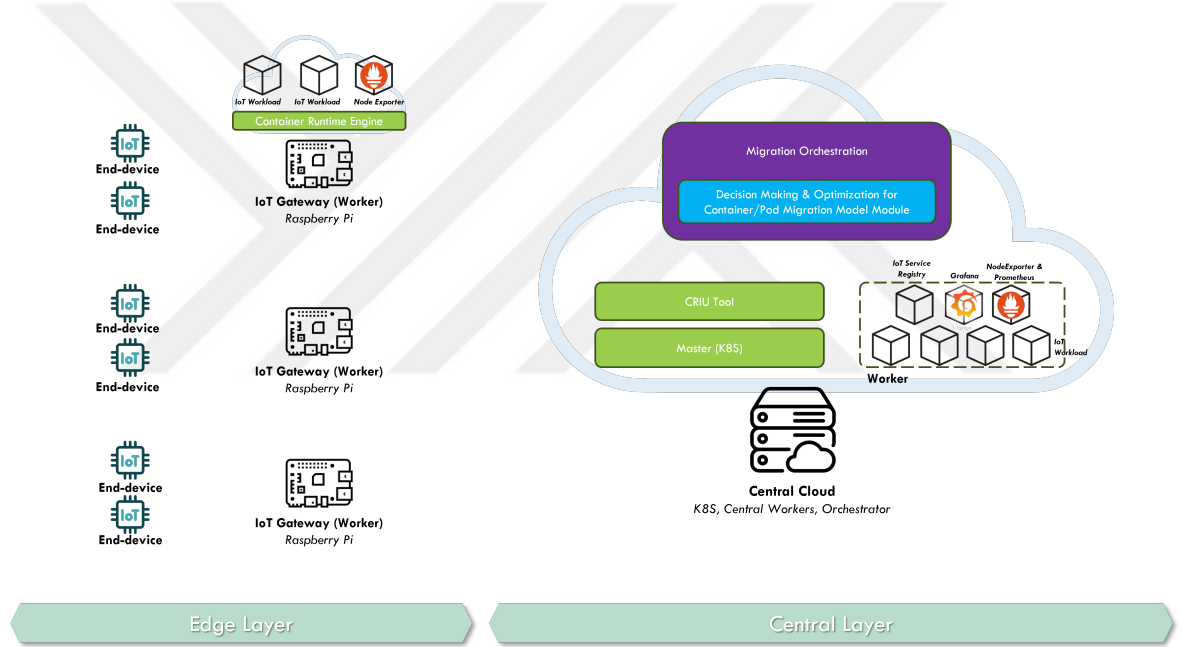


FIGURE 3.1 – High-level Architecture Diagram

## 3.2 Container-based Edge & Central IoT

### 3.2.1 Contanierization

Containers have been adopted as a powerful concept in the field of computing especially in the IoT that projects the considering latter cannot do without efficient and scalable solutions. While have traditional their virtual own machines operating are system heavy kernel, weight containers as are they light weight because they share the host operating kernel. system's This distinction does away with the need for a hypervisor, which is a

layer that is quite heavyweight and is required in case of VMs, which allows containers to run on the host OS directly. The process isolation in containers is done using kernel namespaces whereby each container has its own filesystem, network and process space. Also, control groups allow for accurate resource control over the CPU, memory, and I/O resources by the developers.

A container usually includes only the application, its dependencies in the form of binaries, libraries and tools, and the minimal set of files necessary for the application execution, which makes containers light-weight in terms of storage and start-up time. Despite being meant to be transient, containers can be made to hold state by using features like volumes or by integrating with databases. For this reason that they are versatile, and in conjunction with their light-weight nature, make ideal containers for IoT applications where resource limitations are a factor. Due to their ability to minimize the resources consumption while at the same time providing strong process isolation and flexibility, containers allow IoT edge devices to host several services or applications, conserve power and cope with dynamic workload. Therefore, the concept of containerization is a solution to some of the problems faced in IoT including scalability, low latency and resource consumption and therefore an ideal fit for the future IoT architecture.

### **3.2.2 Edge & Central Kubernetes Architecture**

This research addresses one of the crucial challenges in IoT environments : developing an infrastructure that goes beyond centralized computing to enable dynamic service management and optimization. It achieves this by adopting containerization technology, which is known for its flexibility, lightweight design, and hence a very good choice for IoT applications. In this architecture, IoT gateways operate within a containerized infrastructure, where IoT service workloads are encapsulated as containers. Such a container-based approach will allow for the deployment and control of workloads of IoT service either at the edge or in a centralized manner, based on service requirements like latency reduction, bandwidth optimization, or congestion avoidance.

Container orchestration technologies, especially Kubernetes, shed light on how to manage workloads in containerized form efficiently. Kubernetes, groups containers into pods which allows the orchestration of resources and workloads effectively. The architecture design allows the control plane components — API server, etcd, scheduler,

and controller manager — to run on a central node playing the master role; the edge nodes are specifically worker nodes solely dedicated to running service workloads. This is known as the disaggregated control plane architecture, which fits the edge-central paradigm very well because it pulls all decision-making processes for Kubernetes into the central while offloading the execution of the IoT workloads to the edge nodes. This provides centralization of resource-intensive control operations while conserving resources at the edge but maintaining low-latency, decentralized execution.

The benefits of the container-based framework include, among others, the possibility of dynamically deploying new services and scaling up or down the resources depending on demand; also, workloads can be assigned to specific nodes based on project requirements or hardware capabilities. Furthermore, the stateless nature of containers natively supports workload migration across nodes to dynamically optimize performance and resource utilization. Nevertheless, this work goes beyond the traditional containerized design by considering stateful, closed-loop, continuous workload migrations for different IoT applications. The proposed edge-central architecture enables the dynamic adaptation of IoT services to a change in operational demands and balances the trade-offs effectively between central and edge processing while ensuring strong service continuity and performance. This methodology lays the foundation for efficient, scalable, and adaptable IoT ecosystem designs.

### **3.2.3 CRIU and Live Migration of Workloads**

The proposed edge-central containerized IoT architecture supports the dynamic workloads migration by exploiting CRIU (Checkpoint/Restore in Userspace) to perform seamless live migrations. CRIU is a mechanism designed to dump the state of a running container—including its memory, network connections, opened files, and process tree—and restore it on a different node. Such a feature is vital for supporting scenarios like failover, resource optimization, power saving, and mobility in IoT environments. The live migration process via CRIU consists of several orchestrated steps toward minimal disruption and ensuring a consistent provision of service.

The migration process begins with synchronizing the state of the container on the source and the destination nodes. In this first step, the source node sends the state of the container to the destination, preparing it for the following migration.

Once synchronization is established, the source node proceeds to create a container checkpoint, often called a "dump," which involves the capture of the container's complete runtime state. That includes in-memory data, open file descriptors, existing network connections, and the execution stack. The checkpointing process ensures that the exact state of the container can be restored at some future time on the destination node.

Right after the state is captured, the checkpoint file is transferred to the destination node. This step becomes sensitive to bandwidth and dependent on network efficiency to reduce transfer delays, which have a direct impact on migration downtime. Once the file is successfully received, the destination node confirms that it is ready to proceed with restoring the container.

When it's time for migration, there will be a moment in which the container on the source node is paused in order to temporarily stop everything it was doing and prevent it from making any more modifications to its state. This does guarantee consistency between the two environments of the source and destination nodes. Meanwhile, the container is restarted at the destination node using a checkpoint file. In other words, this restore guarantees exactly the same state—say, some established connections or ongoing computation—so that it goes on without interruption.

After the successful restoration, the destination container is verified and reactivated to ensure that it is functioning properly in the new environment. Once the source node receives an acknowledgement from the destination node that the container is functioning properly, it decommissions and deletes the source container to free up resources and prevent duplication. This marks the end of the migration process, and the workload is now fully functional on the destination node.

With its ability to handle complex container states with minimal service disruption, CRIU stands out as an ideal candidate for proposed architecture. Each of these steps is integrated within the Kubernetes-based orchestration system, automated through the decision-making process of the fuzzy logic-based migration. Combining these features of CRIU with real-time metrics across hardware, network, container, and service layers, the architecture assures optimized migrations tailored to dynamic IoT system needs. This workflow only reduces the downtime but also increases the flexibility and productivity of the edge-central IoT ecosystem, making it resilient toward various operational challenges.

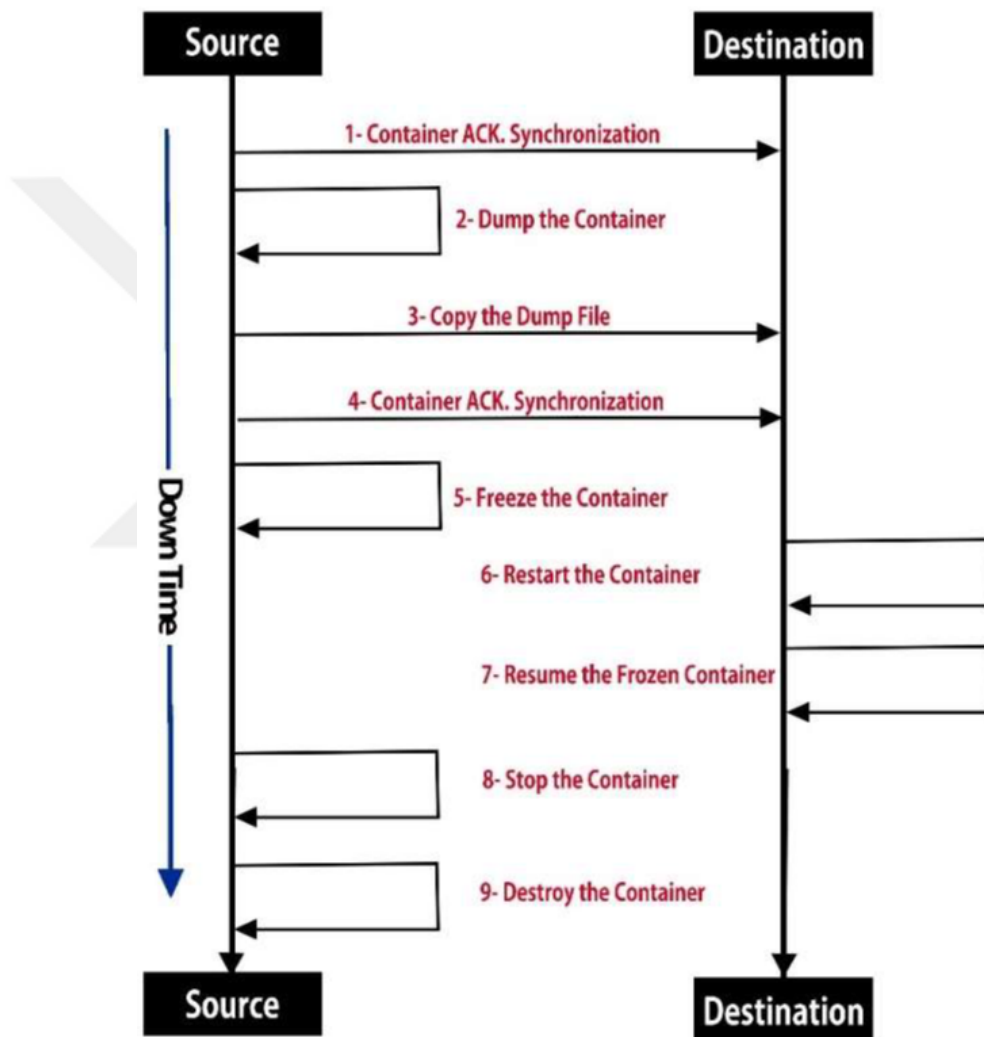


FIGURE 3.2 – CRIU based Checkpoint Flow

### 3.3 IoT Service Migration Scenarios

In the proposed edge-central containerized IoT architecture, live migration plays a pivotal role in maintaining service continuity, optimizing resource utilization, and adapting to dynamic operational conditions. Several scenarios have been identified to showcase the robustness and adaptability of the system, each reflecting real-world challenges in IoT environments. The scenarios are using a fuzzy-logic-based decision-making model, which considers various metrics at hardware, network, container/pod, and IoT service levels to dynamically decide and negotiate the most appropriate migration strategies. Details of the each scenario and its flow will be identified in the experiment section.

1. **Container/Pod or Node Failover** : IoT systems require strong failover mechanisms that ensure the reliability of the systems. In the event of a failure of a container or pod, or whenever a node becomes unavailable, the decision model quickly identifies the most appropriate replacement node using predefined thresholds and metrics such as resource availability, network congestion risk, and workload compatibility. Then, the workload that has failed is effectively migrated to the selected horizontal or vertical node by applying CRIU-based live migration in order to minimize downtime and maximize continuity of services.
2. **Resource Optimization** : IoT environments face dynamically changing workloads and resource demands. The decision model considers hardware-level metrics like CPU and memory utilization, network parameters like latency and bandwidth, and service-level metrics for the most efficient workload redistribution among the horizontal and vertical nodes. For example, heavily loaded nodes can offload some of their workload to underutilized nodes, thus reducing bottlenecks and improving the overall performance of the system. This proactive optimization increases the operational lifetime of edge devices by preventing resource overloading and ensuring balanced utilization across the network.
3. **New Workload Deployment** : The integration of new services or applications into the IoT ecosystem requires accurate resource allocation in order to ensure a harmonious deployment. These two aforementioned roles of the decision-making framework evaluate the current usage of resources by nodes, addressing aspects such as distance to IoT devices, available capacity, or potential service interdependencies. Hence, by determining the optimal node for the new arriving workload, the system may avoid potential conflicts and achieve optimal resource allocation.

Thus, its dynamic scaling of operations works based on very fast implementation of additional functionalities in response to user requests or changing environmental conditions.

4. **Energy-Saving** : Energy efficiency in IoT systems, mainly for edge devices working in low power, becomes prominent. This decision model involves the metrics of power consumption and energy efficiency at the hardware level to dynamically migrate the workload to nodes that will be able to execute at less cost in terms of power—say, consolidation of underutilized nodes such that some of the nodes may go into low-power states or may temporarily be shut down. This will reduce the overall energy footprint of the IoT system, yet preserve the quality of service—contributing to the sustainability of edge-based IoT deployments.
5. **Thing Mobility** : Mobility introduces special challenges in IoT environments, where devices may move between IoT gateways. Whenever an IoT device connects to a new gateway, the associated workloads have to follow to ensure continuity. The decision model identifies the target gateway and orchestrates the migration of workloads to it, considering factors like latency, network stability, and resource compatibility. This scenario evidences the dynamic nature of the proposed architecture, as it is able to adapt to changes in the physical location of IoT devices without interrupting service delivery. This capability is of particular importance for applications like connected vehicles or mobile public health monitoring systems.

These migration scenarios are central to the study objectives and demonstrate how the proposed architecture can meet common IoT challenges with intelligent decision-making and orchestration. The research study implements these scenarios in testing and achieving a scalable, resilient, and efficient framework for the management of IoT workloads in an edge-central containerized environment.

### 3.4 Service-based IoT Communication

One of the common architectural paradigms in modern computing, as illustrated by frameworks including the 5G Core Network and IT workloads, is the Service-Based Architecture (SBA). Within this architecture, software components interact with each other based on web services (REST APIs) are often adopted for effective integration and smooth communication within both the internal and external networks. That in-

herent simplicity, adaptability, and scalability make this model a bedrock upon which modern applications are designed. When coupled with containerization—an approach aligned with microservice architecture principles that encapsulates individual software processes into isolated, lightweight units—SBA and containerization complement each other in creating a modular, scalable system architecture.

Containerized IoT architectures contain microservices/workloads, each designed to implement a particular function and possessing unique attributes and purpose. A central service registry is, therefore, in many cases, used to provide easy communication and coordination between these services. It also provides the registry, a centralized storage of service-related metadata—attributes, parameters, and operational states—that can help achieve proper discovery and communication between services. This architecture implements the consumer-producer pattern, where services (consumers) discover and involve other services (producers) dynamically based on their current needs.

Initially, services register into the central registry and continue "heartbeat" communications with regular intervals to validate the availability of these services. Moreover, if there is any updates on service, service needs to be update its "profile" on registry, thus ensuring a current and complete system-wide view. A consumer can subscribe to a service producer for periodic information about related service for any purpose in scope of service's objective. This functionality can support periodic interactions with customized intervals in line with the objectives of the IoT architecture. As an example, an IoT workload responsible for monitoring environmental sensors might subscribe to updates from a service that aggregates regional sensor data, thereby supporting responsive, data-driven decision-making.

This study follows the SBA paradigm in order to show efficient management of IoT services in the containerized edge-central architecture. The central IoT service registry is an integral part of the project's framework, acting as a repository for service-related metadata and as a source providing metrics to feed migration decisions. With the integration of this registry in the IoT service layer, the architecture not only guarantees continuity of operations but also enhances the decision process for dynamic workload migration. In this way, the system can further optimize performance, resource utilization, and service reliability, pointing out the synergy of SBA and containerized IoT architectures in achieving scalable and adaptive solutions.

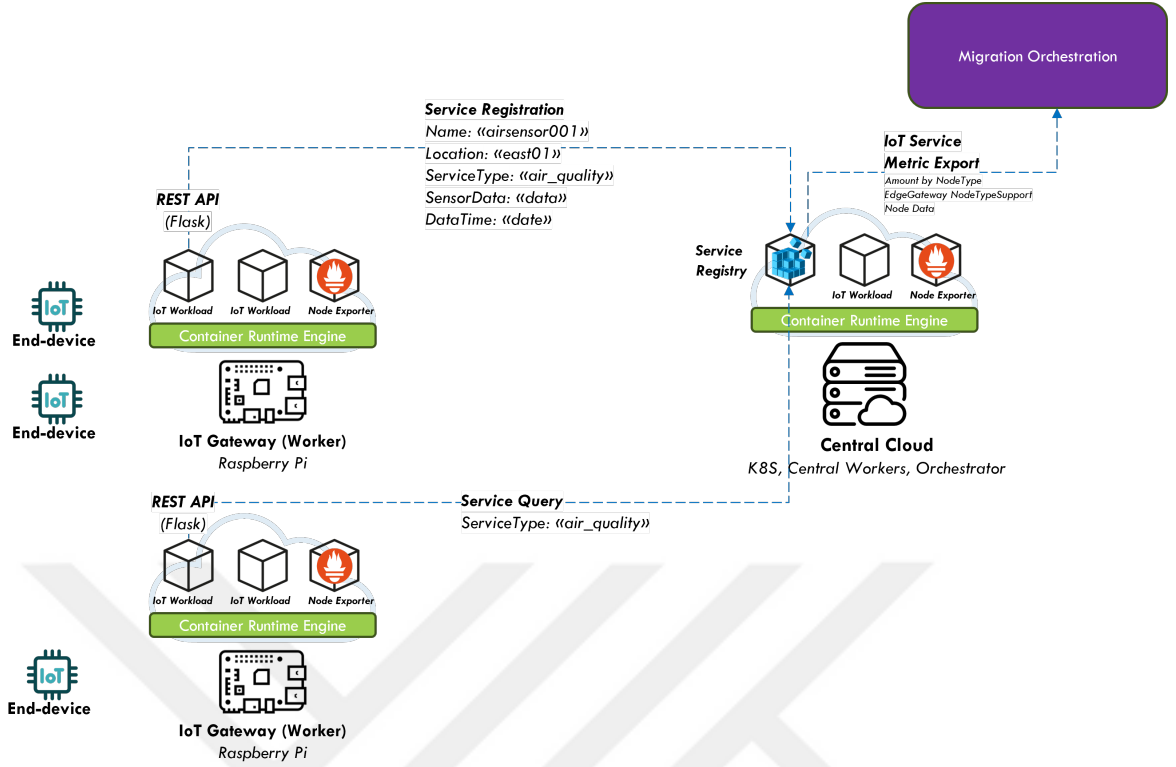


FIGURE 3.3 – Service based Architecture &amp; Service Registry Diagram

### 3.5 Metrics & Metric Management

In the proposed edge-central containerized IoT architecture, live migration decisions rely on a comprehensive set of metrics spanning hardware, network, container/pod, and IoT service levels. These metrics are essential in the evaluation of the system state and the optimal migration strategy for efficient resource utilization and low latency of the services.

With the integration of these metrics within the fuzzy logic decision-making module, the system becomes able to acquire a multi-dimensional view of the environment for context-aware, efficient migration. In other words, hardware metrics ensure resource availability and reliability, network metrics maintain communication quality, container-level metrics validate workload compatibility, and IoT service metrics align migration with application-specific needs. This holistic approach ensures that migrations are not only technically feasible but also optimized for performance, energy efficiency, and service quality, enabling a resilient and adaptive edge-central IoT ecosystem.

#### Hardware-Level Metrics

1. CPU Usage (%) : This shows the load of the processor in each node. It helps in locating nodes that are underused or overloaded. High CPU usage may require redistribution of the workload, to prevent bottlenecks and ensure smooth operation.
2. Memory Usage (%) : Memory availability is one of the most important aspects of running containerized workloads. Monitoring memory usage helps avoid overloading nodes and ensures sufficient capacity for migrating workloads.
3. Storage Utilization (%) : Having limited storage capacity might impact the possibility of hosting additional workloads. This metric ensures that the destination nodes have enough storage resources for the transfer of data and container images.
4. Thermal Status / Temperature (°C) : Overheating can reduce hardware lifespan and lead to failures. Temperature monitoring helps in migration decisions to balance thermal loads across nodes and avoid hardware degradation.
5. Battery Level (%) : This metric is very important for battery-powered edge devices in order to determine if a node can handle extra workloads or should offload the tasks to save energy.
6. Power Consumption (W) : Monitoring power consumption supports energy-saving migration scenarios, optimizing workload placement on nodes with lower power usage for sustainability.

### **Network-Level Metrics**

1. Bandwidth Utilization (%) : High bandwidth usage may indicate network congestion. This measure is used to identify nodes with enough network bandwidth for both migration and steady-state communication.
2. Data Throughput (Mbps) : Throughput measures the network's data transfer rate. Higher throughput indicates better performance and suitability for hosting additional workloads.
3. Network Latency (ms) : Low latency is critical for time-sensitive IoT applications. Migration decisions consider latency to maintain service quality, especially for edge nodes serving real-time devices.
4. Packet Loss Rate (%) : The packet loss can also disturb service communication and data integrity. The work load should be migrated to the nodes with lower packet loss rates.
5. Jitter (ms) : Jitter refers to variations in packet arrival times and has an impact on

service reliability. Stable nodes with low jitter are preferred for hosting migrated workloads.

6. Overall Network Congestion Level (%) : This metric provides a higher-level view of the health of a network and assists in making decisions about avoiding nodes in highly congested regions.

### **Container/Pod-Level Metrics**

1. Pod CPU Usage (%) : The processing demand of individual pods helps determine which workloads may need migration to balance computational loads across nodes.
2. Pod Memory Utilization (%) : Memory utilization at the pod level should be monitored to ensure workload transfer does not overpower available resources on the target node.
3. Pod Storage/Disk Utilization (%) : This metric ensures that the container data requirements are met during migration, ensuring no issues with storage constraints.

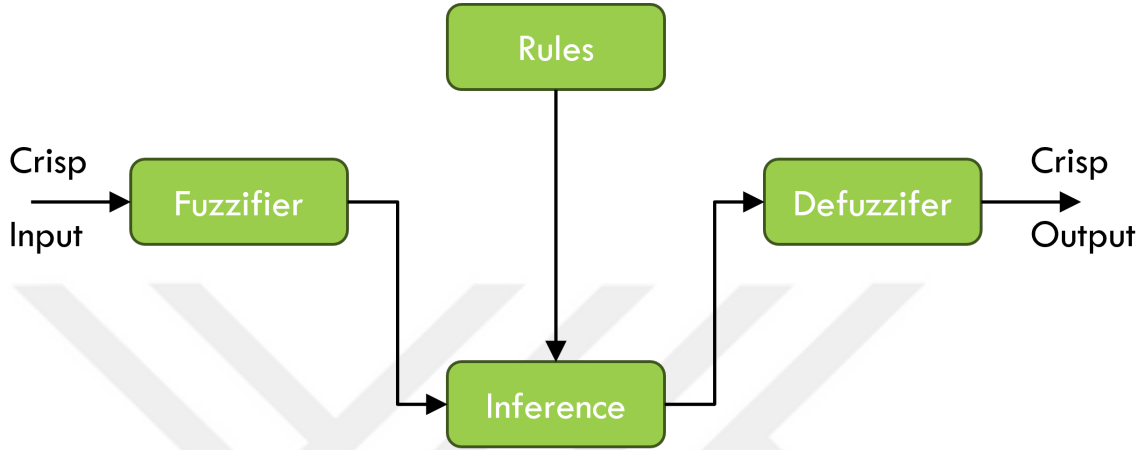
### **IoT Service-Level Metrics**

1. End-Device Type : IoT devices have diverse computational and network requirements ; thus, knowledge of the device type will help optimize workload placement.
2. Service Priority : Not all services are equally important. Priority levels set the migration decision model in motion to ensure high-priority services are preserved with minimal disruption.
3. Device Mobility : Mobile IoT devices impose the necessity of dynamic workload migration, since such devices are attached to different edge gateways. This metric ensures continuity of service by adapting workloads with device location changes.
4. Device/Node Location : Proximity to the end device is important for low-latency services. Location metrics help to choose the nearest and most suitable node for migration.

## **3.6 Fuzzy Logic based Migration Decision Making**

Fuzzy logic inference systems are computational models designed to facilitate approximate reasoning in conditions of uncertainty, imprecision, and partial truth—characteristics

typical of real-world systems such as robotics, control systems, and adaptive resource allocation. Two of the most dominant architectures within this paradigm are the **Mamdani-type** and **Sugeno-type** fuzzy inference systems (FIS). While both rely on linguistic rule-based models, they differ significantly in their treatment of consequent functions and defuzzification processes.



Structure of a  
FIGURE 3.4 – Fuzzy Inference System

The *Mamdani FIS*, developed by Ebrahim Mamdani in 1975, is grounded in Zadeh's foundational theory of fuzzy sets. It is especially suited for encoding expert knowledge in a manner that is intuitively interpretable by humans. Both antecedents and consequents in Mamdani-type rules are expressed using linguistic variables and associated membership functions. A typical Mamdani fuzzy rule is structured as follows :

$$IF \ x \text{ is } A \ AND \ y \text{ is } B \ THEN \ z \text{ is } C$$

where  $A$ ,  $B$ , and  $C$  are fuzzy sets defined over their respective universes of discourse.

The *Sugeno FIS*, introduced by Takagi, Sugeno, and Kang in the 1980s, retains the same structure for antecedents but introduces a key difference in the consequent part : it utilizes crisp mathematical functions rather than fuzzy sets. A typical Sugeno fuzzy rule can be represented as :

$$IF \ x \text{ is } A \ AND \ y \text{ is } B \ THEN \ z = f(x, y)$$

where  $f(x, y)$  is usually a linear or constant function. This design improves computational efficiency and compatibility with optimization and control paradigms.

Both Mamdani and Sugeno systems share a common sequence of computational steps :

**fuzzification, rule evaluation, aggregation, and defuzzification.** However, the internal logic and execution differ in meaningful ways.

Mamdani based FIS can be summarized with below items :

- **Fuzzification** : Converts crisp inputs into degrees of membership within relevant fuzzy sets.
- **Rule Evaluation** : Applies fuzzy operators (e.g., *minimum* for *AND*) to compute the firing strength of each rule.
- **Implication and Aggregation** : Constructs and combines output fuzzy sets using methods such as minimum or product.
- **Defuzzification** : Employs techniques such as centroid or bisector methods to transform the aggregated fuzzy set into a single crisp output.
- **Remarks** : The system is highly interpretable but introduces computational overhead during defuzzification due to the integration over fuzzy output sets.

Sugeno based FIS can be summarized with below items :

- **Fuzzification and Rule Evaluation** : Identical to the Mamdani system.
- **Consequents** : Represented by crisp mathematical functions rather than fuzzy sets.
- **Output Aggregation** : Final output is computed as a weighted average of the rule outputs :

$$z = \frac{\sum_i w_i f_i(x)}{\sum_i w_i}$$

where  $w_i$  denotes the firing strength of the  $i$ -th rule.

- **Remarks** : Offers high-speed computation and is well-suited for integration into adaptive and learning systems. However, this comes at the cost of reduced linguistic interpretability.

Fuzzy logic provides a robust framework to handle uncertainty and imprecision in decision-making, especially in dynamic and heterogeneous environments like IoT ecosystems. In this project, fuzzy logic is employed to decide whether container workloads should be migrated between nodes (edge or central) based on a variety of metrics at the hardware, network, container, and service levels. The system is designed to evaluate multiple inputs continuously, map these inputs to fuzzy sets, infer relationships using predefined rules, and produce crisp outputs to guide migration actions.

The essence of using fuzzy logic lies in its ability to handle overlapping input conditions (e.g., high CPU usage and moderate memory usage) and provide a nuanced decision

rather than a binary yes/no response. This is particularly valuable in IoT environments where metrics often fluctuate and precise thresholds may not capture the true state of the system.

### 3.6.1 Multidimensional Input Metrics

Mamdani-based fuzzy logic model begins by defining the input metrics, which are derived from four fundamental dimensions : hardware, network, container/pod, and IoT service levels. These metrics capture the real-time state of the system to inform migration decisions.

#### Hardware-Level Metrics :

$$M_H = \{M_{H,CPU}, M_{H,Memory}, M_{H,Storage}, M_{H,Power}, M_{H,Battery}\}$$

Hardware resources are critical for maintaining the performance and availability of IoT workloads. These metrics directly influence migration decisions, especially under high load or power constraints.

CPU Usage ( $M_{H,CPU}$ ) : Represents the percentage of processor capacity in use.

Memory Usage ( $M_{H,Memory}$ ) : Measures the utilized portion of available memory.

Storage/Disk Utilization ( $M_{H,Storage}$ ) : Represents the percentage of storage or disk capacity currently utilized on the hardware node. It helps assess whether the storage resource is a bottleneck for the workloads.

Power Consumption ( $M_{H,Power}$ ) : Indicates the wattage consumed by the node.

Battery Level ( $M_{H,Battery}$ ) : Tracks the energy remaining in battery-operated nodes.

#### Network-Level Metrics :

$$M_N = \{M_{N,BW}, M_{N,Lat}\}$$

Network performance impacts the feasibility of migrating workloads

Bandwidth Utilization ( $M_{N,BW}$ ) : Measures the proportion of network capacity used.

Network Latency ( $M_{N,Lat}$ ) : Indicates the delay in data transmission, critical for real-time IoT workloads.

### Pod/Container-Level Metrics :

$$M_P = \{M_{P,CPU}, M_{P,Memory}, M_{P,Storage}\}$$

Pod-specific metrics focus on individual workload resource usage

Pod CPU Usage ( $M_{P,CPU}$ ) : Reflects the CPU consumption of a specific container/pod.

Pod Memory Usage ( $M_{P,Memory}$ ) : Measures the percentage of memory utilized by a specific pod or container. It reflects the workload's memory intensity and helps assess whether memory constraints affect performance.

Pod Storage/Disk Utilization ( $M_{P,Storage}$ ) : Represents the percentage of storage capacity utilized by the container or pod. This metric helps identify containers that are storage-intensive and may benefit from migration to a node with more available storage.

### IoT Service-Level Metrics :

$$M_S = \{M_{S,Priority}, M_{S,Mobility}, M_{S,DeviceType}, M_{S,Location}\}$$

IoT service metrics capture the operational priorities of services. Each metric provides a crisp numerical value, which serves as the input to the fuzzy logic system. These values has to be continuously collected and observed via regarding listening approaches.

Service Priority ( $M_{S,Priority}$ ) : Reflects the criticality of the service (e.g., high-priority medical devices).

Device Mobility ( $M_{S,Mobility}$ ) : Indicates whether the associated IoT device is static or moving.

End Device Type ( $M_{S,DeviceType}$ ) : Categorizes the type of IoT device (e.g., sensor, gateway, etc.) connected to the system. This metric helps prioritize devices based on their importance or role.

Device/Node Location ( $M_{S,Location}$ ) : Indicates the geographical or logical location of the IoT device or node. This metric is critical for evaluating migration feasibility based on proximity or latency considerations.

### 3.6.2 Fuzzification

Fuzzification is the process of converting crisp numerical values into fuzzy linguistic terms. This step translates raw data into a format that is meaningful for reasoning.

Membership functions define the degree to which an input value belongs to a fuzzy set. Each input metric is associated with one or more linguistic terms, such as "low," "medium," and "high." Mathematically, a membership function  $\mu(x)$  maps a crisp value  $x$  to the interval  $[0, 1]$ . Each fuzzy set has a distinct membership function tailored to the domain of the input metric.

**Membership Function for "Low" Pod Memory Usage :**

$$\mu_{\text{Low}}(x) = \begin{cases} 1 & \text{if } x \leq 20, \\ 1 - \frac{x-20}{30} & \text{if } 20 < x \leq 50, \\ 0 & \text{if } x > 50. \end{cases}$$

**Membership Function for "Medium" Pod Memory Usage :**

$$\mu_{\text{Medium}}(x) = \begin{cases} \frac{x-20}{30} & \text{if } 20 < x \leq 50, \\ 1 - \frac{x-50}{30} & \text{if } 50 < x \leq 80, \\ 0 & \text{if } x \leq 20 \text{ or } x > 80. \end{cases}$$

**Membership Function for "High" Pod Memory Usage :**

$$\mu_{\text{High}}(x) = \begin{cases} \frac{x-50}{30} & \text{if } 50 < x \leq 80, \\ 1 & \text{if } x > 80, \\ 0 & \text{if } x \leq 50. \end{cases}$$

**Membership Function "Near" IoT Device/Gateway Location :**

$$\mu_{\text{Near}}(x) = \begin{cases} 1 & \text{if } x \leq 30, \\ 1 - \frac{x-30}{20} & \text{if } 30 < x \leq 50, \\ 0 & \text{if } x > 50. \end{cases}$$

### Membership Function "Far" IoT Device/Gateway Location :

$$\mu_{\text{Far}}(x) = \begin{cases} \frac{x-30}{20} & \text{if } 30 < x \leq 50, \\ 1 & \text{if } x > 50, \\ 0 & \text{if } x \leq 30. \end{cases}$$

The fuzzification step allows inputs to be evaluated across overlapping sets. For instance, if  $M_{P,\text{Memory}} = 35\%$  it may partially belong to "low" ( $\mu_{\text{Low}} = 0.5$ ) and "medium" ( $\mu_{\text{Medium}} = 0.5$ ) enabling nuanced decision-making.

### 3.6.3 Rule Base Structure

Fuzzy rules form the core of the decision-making system. These rules capture expert knowledge and encode it in an declarative format.

#### General Rule Structure :

$$R_i : \text{IF } (x_1 \text{ is } A_1) \wedge (x_2 \text{ is } A_2) \wedge \dots \wedge (x_m \text{ is } A_m) \text{ THEN } (y \text{ is } B)$$

where :

$x_1, x_2, \dots, x_m$  are inputs,

$A_1, A_2, \dots, A_m$  re fuzzy sets for the inputs,

$y$  is the output variable,

$B$  is the fuzzy set for the output.

#### Specific Rules :

1. Failover Scenario :

$$R_1 : \text{IF } (M_{H,\text{CPU}} \text{ is High}) \wedge (M_{H,\text{Mem}} \text{ is High}) \text{ THEN } (Migration \text{ is Immediate})$$

2. Energy-Saving Scenario :

$$R_2 : \text{IF } (M_{H,\text{Power}} \text{ is High}) \wedge (M_{H,\text{Battery}} \text{ is Low}) \text{ THEN } (Migration \text{ is Immediate})$$

3. Mobility Scenario :

$$R_3 : \text{IF } (M_{S,\text{Mobility}} \text{ is Mobile}) \text{ THEN } (Migration \text{ is Immediate})$$

These rules are aggregated into a rule base, allowing the system to evaluate multiple conditions simultaneously.

### 3.6.4 Inference

Inference is the process of applying the fuzzy rules to determine the fuzzy output. The AND operator is used to evaluate the rule antecedents.

**Rule Activation Degree :**

$$\mu_{\text{Rule}} = \min(\mu_{A_1}(x_1), \mu_{A_2}(x_2), \dots, \mu_{A_m}(x_m))$$

**Example Calculation :**

$$\mu_{\text{Rule}} = \min(\mu_{\text{High, Power}}(x), \mu_{\text{Low, Battery}}(x))$$

If :

$$\mu_{\text{High, Power}}(x) = 0.7, \quad \mu_{\text{Low, Battery}}(x) = 0.8,$$

Then :

$$\mu_{\text{Rule}} = \min(0.7, 0.8) = 0.7$$

This value (0.7) is propagated to the output fuzzy set.

### 3.6.5 Defuzzification

The defuzzification step converts the fuzzy output into a crisp decision. The centroid method is commonly used.

**Centroid Method Formula :**

$$y^* = \frac{\int y \cdot \mu(y) dy}{\int \mu(y) dy}$$

where :

$y$  is the output variable (e.g., migration decision),

$\mu(y)$  is the membership degree.

### 3.6.6 Continuous Evaluation

In a dynamic container-based IoT environment, this entire process is repeated continuously. Metrics are updated periodically, decisions are re-evaluated, and actions are executed as needed. This ensures the system adapts to changes in real-time, optimizing performance and resource utilization.

## 3.7 Migration Orchestration

Once migration decisions have been made by the fuzzy logic-based decision-making module, a service orchestration module plays a key role in realizing these decisions within the containerized IoT architecture. The fuzzy logic component evaluates, in real time, metrics from the hardware, network, container/pod, and IoT service layers to determine the best migration strategy. Meanwhile, the service orchestration component assures the seamless execution of the decided actions by interacting with the Kubernetes cluster and the tool CRIU. This way, integration brings resilience, efficiency, and adaptability to containerized workload management across the edge-central continuum.

The service orchestration module acts as an execution layer that takes high-level migration decisions from the fuzzy logic module and translates them into actionable tasks within the IoT environment. It interacts with the K8S API for efficient management of pods and container states, whether it is an edge-to-edge horizontal migration or an edge-to-central vertical migration. It initiates CRIU-based live migration in order to seamlessly transfer workloads with minimal downtime and data loss. This requires the tight collaboration of the source and destination nodes on the container state checkpoints, resource allocation, and network settings.

The orchestration module also ensures checks of compatibility and handles errors during migrations. For example, before starting the migration, it is ensured that the destination node has available resources—CPU, memory, and storage—and network bandwidth. On occurrence of an error during migration, say resource unavailability or failure of connection, the orchestration module performs a predefined number of retries for this operation. In the case where the failure remains, the orchestration initiates different migration scenarios; this might redistribute the workloads among the other nodes that are qualified, based on a changed priority score of this particular IoT

environment.

The orchestration module’s critical functionalities include scalability and coordination among multiple nodes. The system enables parallel migration of several containers or pods to different nodes, following the recommendations from the fuzzy logic model. In the case of a failover scenario where many pods of a faulty node must be migrated, the module will guarantee that all affected services are migrated to suitable nodes while preserving their dependencies and service-level agreements (SLAs). Similarly, during resource optimization, it will orchestrate the redistribution of workload across nodes, minimizing bottlenecks and balancing resource utilization.

Another significant responsibility of the orchestration module is state management. The module maintains a global, up-to-date state database of all nodes, including their resource usage, network metrics, and currently hosted workloads. This stateful design allows the module to be dynamic in reaction to environmental changes, such as when an IoT device changes its association to another gateway or resource demand increases with time and new workload needs deployment. The metrics-to-node and-workload mapping ensures that actions are accurate and context-aware.

Finally, the orchestration module is designed with extensibility in mind. It provides interfaces to easily integrate additional metrics or parameters as needed, ensuring that the architecture can evolve to meet new challenges. For instance, it is within the possibilities of developers to integrate metrics related to advanced IoT service-level parameters or to improve compatibility with newly appearing migration tools other than CRIU. Subsections describe migration orchestration workflows and requirements one-by-one. It can be noted that for practical reference, real-life solutions has been utilized in the workflows (e.g. Prometheus, CRIU).

### **3.7.1 Decision Making & Migration Orchestration Workflow**

The migration decision and orchestration system is designed using a modular architectural model comprising six distinct but interrelated modules, each has been referenced within a Python-based framework. These modules collectively address specific functional requirements within the broader scope of adaptive containerized workload management in edge-cloud environments.

- **Fuzzy Logic System Module** : This module employs a fuzzy inference engine to emulate decision-making under uncertainty. It transforms heterogeneous input metrics into migration decisions through a rule-based control mechanism that enables nuanced reasoning across diverse system conditions.
- **Metrics Collection Module** : Leveraging Prometheus as the underlying telemetry infrastructure, this module systematically gathers time-series data across multiple operational domains, including hardware performance, network health, Kubernetes resource utilization, and IoT service-level indicators.
- **CRIU Integration Module** : This module facilitates the checkpointing and restoration of containers by interfacing with the Checkpoint/Restore In Userspace (CRIU) utility. It enables stateful container migration with minimal disruption to ongoing services.
- **Worker Node Tracking Module** : Responsible for the continuous monitoring of worker node conditions, this module periodically updates the system's internal state representation to reflect current resource availability and performance metrics throughout the distributed infrastructure.
- **Migration Manager Module** : As the core decision-execution engine, this module evaluates migration urgency based on outputs from the fuzzy logic system, selects optimal target nodes through multi-criteria optimization, and initiates the live migration process accordingly.
- **Main Execution Module** : This module orchestrates the concurrent execution of monitoring and migration tasks using lightweight threading. It ensures uninterrupted system operation and real-time responsiveness within dynamic and resource-constrained environments.

### 3.7.1.1 Fuzzy Logic System Module & Fuzzy Logic Algorithms

As explained previously, the fuzzy logic system acts like an intelligent decision-making mechanism, which is designed to process the different input variables and come up with a migration decision. The first step is the definition of the input variables—antecedents—such as CPU usage, memory consumption, and network latency, representing the most important performance indicators of the system. Each variable has a given range associated with it, and membership functions are developed to partition the ranges into overlapping groups such as "low," "medium," and "high." These groups enable the

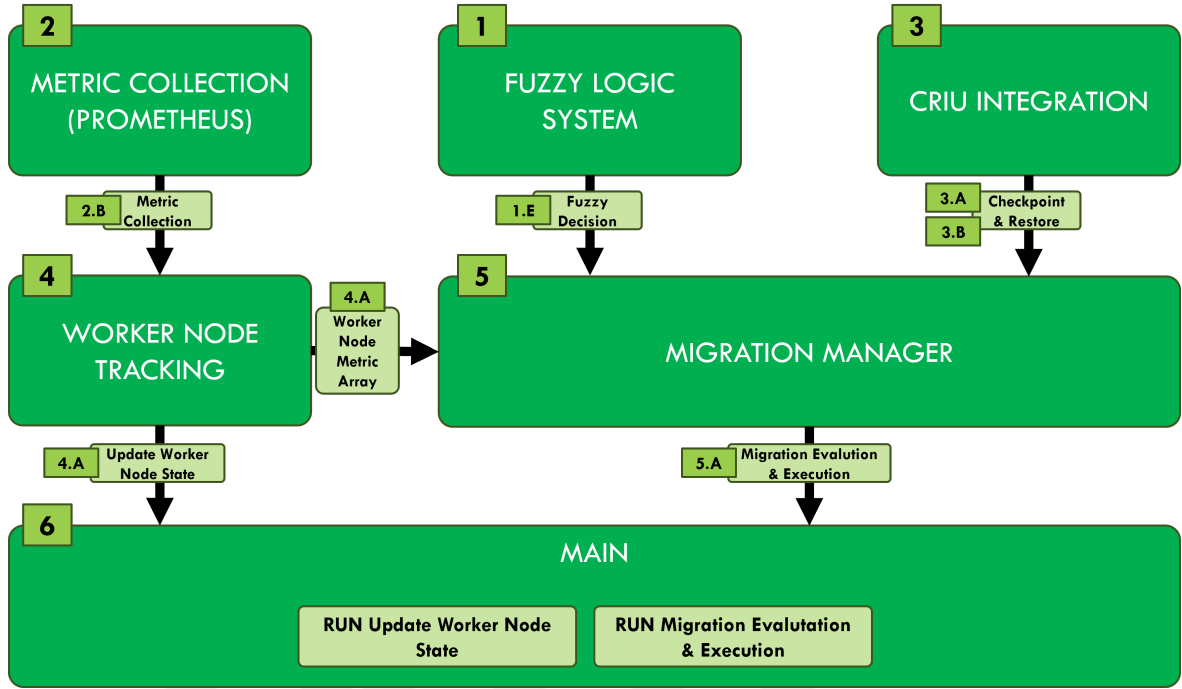


FIGURE 3.5 – Structure of Decision Making & Migration Orchestration System

system to change smoothly between states so that small changes in metrics do not result in abrupt changes in decisions.

The system also fuzzifies a consequential variable called the Migration Decision, which enables it to categorize possible migration actions into "none," "moderate," and "immediate." A human-interpretable mapping from multidimensional system states to operational actions is offered by this concept.

After developing membership functions, the system follows a rule-based approach to reach decisions. Rules are defined, which describe the relationship between inputs and the expected output—say, "IF CPU usage is poor AND memory usage is poor, THEN migration decision is immediate." The fuzzy inference engine, at runtime, takes the real-time data from the system. It computes the degree to which each rule is satisfied based on current input values. These evaluations are composed using fuzzy operators such as AND, OR, and NOT, which provides a fuzzy output set.

The fuzzy output set is then processed in a defuzzification step, which translates it into a clear numerical value, such as a migration decision score. This score represents the urgency of migration and is compared with set thresholds to determine what should be done next. It can respond to many inputs simultaneously, as well as to dynamically changing conditions—features that make it ideal for the IoT environment.

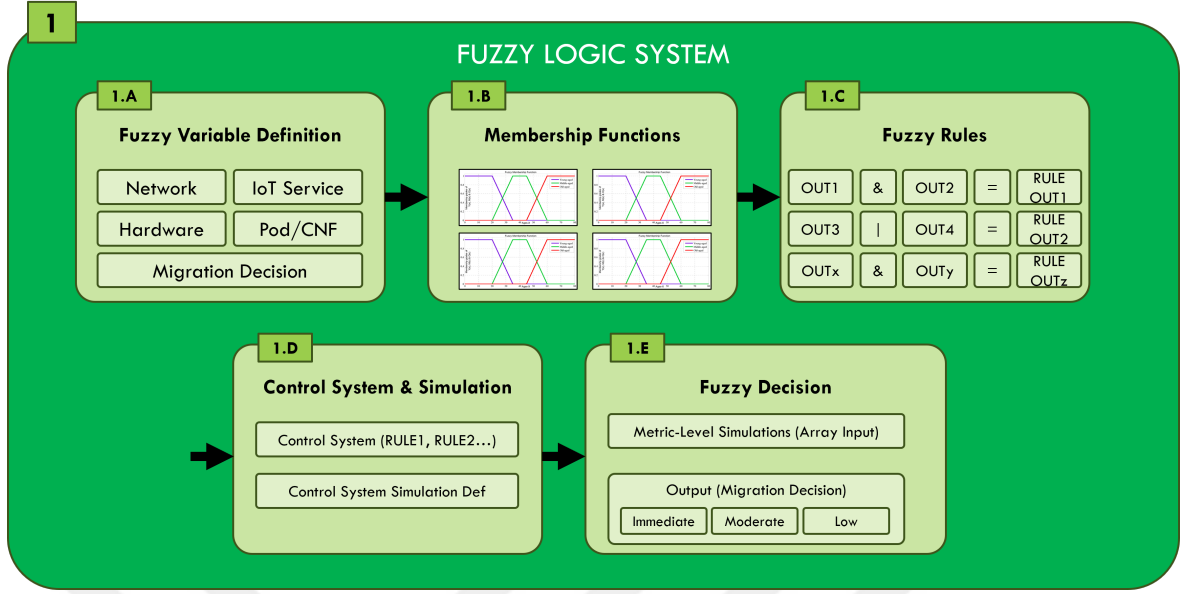


FIGURE 3.6 – Fuzzy Logic Module

A list of current metric values and an actual monitoring output (from Prometheus, for example) are supplied as the system's input. Because the decision engine is never connected to either the migration actuation logic or the metric gathering subsystem, this format offers modularity. As a result, the fuzzy logic module is a stand-alone, state-less part that may be added, removed, or reused without compromising the architecture around it.

Additionally, the fuzzy engine uses an interpretable, non-parametric decision model that eliminates the need for training data-intensive, uninterpretable black-box learning procedures. This feature is especially helpful in situations involving mission-critical edge computing when deterministic and transparent behavior is necessary.

### Mamdani Fuzzy Inference System

Mamdani FIS employs **fuzzy rule-based reasoning** where the output is in the form of a **fuzzy set**, and a **defuzzification step** is required to obtain a crisp migration decision. *centroid method* is employed for defuzzification, which computes the center of gravity of the combined membership functions to obtain the final decision, which is mathematically represented as :

$$D_{\text{Mamdani}} = \frac{\sum_i \mu_i(x) \cdot x_i}{\sum_i \mu_i(x)} \quad (3.1)$$

where  $D_{\text{Mamdani}}$  is the final migration decision,  $\mu_i(x)$  is the membership value of an individual rule output, and  $x_i$  is the corresponding crisp value. It is an interpretability-aware method and highly similar to human decision-making that can be efficiently used in applications where **transparency in decision-making** is of the highest priority.

### Sugeno Fuzzy Inference System

The Sugeno FIS employs a **function-based output approach**, where the fuzzy rules map inputs to **constant values or linear equations**. Instead of defuzzification, the Sugeno model applies **weighted average calculations**, where the final migration decision is computed as :

$$D_{\text{Sugeno}} = \frac{\sum_i w_i f_i(x)}{\sum_i w_i} \quad (3.2)$$

where  $D_{\text{Sugeno}}$  is the final migration decision,  $w_i$  is the activation weight of the  $i$ -th rule, and  $f_i(x)$  is a polynomial function of input variables. For instance, in our containerized IoT environment, we can specify a Sugeno rule as :

$$f_i(x) = a \cdot \text{CPU\_Usage} + b \cdot \text{Memory\_Usage} + c \quad (3.3)$$

where  $a, b, c$  are pre-computed coefficients based on system optimization. This is an efficient computation method that reduces processing time, hence more suitable for real-time and adaptive decision-making in large-scale, latency-constrained systems such as containerized IoT networks.

#### 3.7.1.2 Metric Collection & Aggregation Module

For collecting and providing real-time metrics from system nodes, Prometheus has been referenced as a source. First, one needs to define what metrics are required ; examples of some are CPU utilization, memory usage, and network bandwidth. Each of these metrics corresponds to a respective query in respecting language (e.g. Prometheus' query language, PromQL). The metrics collection system sends a query to source node (e.g. Prometheus' HTTP API), specifying which metric is desired and on which node or instance it should apply. For example, the average CPU usage of a given node over the last minute might be requested. Each metric from each node collected via exporter

services.

Source exporter returns the requested information in a time-series value format. The metrics collection system parses this information and pulls out the most recent values to be fed into the fuzzy logic system. Where there are multiple nodes or metrics being monitored, this system loops over each of the queries, ensuring a collection of all data.

This is done at fixed intervals to provide a run-time monitoring capability. The metrics thus collected are stored in an organized manner so that other components, like the worker state tracker and migration manager, can access them as required. This real-time data acquisition framework ensures that decisions are always based on the latest state of the system, allowing for immediate and accurate responses to dynamic situations.

In scope of how the metric collection and aggregation module operates as; a structured data extraction mechanism is used to retrieve node-specific metric values from source exporter as part of the metric collection and aggregation module's overall operation. The algorithm accepts the name of the related metric of interest and the identifier of the node to be monitored as two arguments. The dynamically produced query that is then sent to the metric exporter node server over an available connection interface is built upon these parameters.

The query string is created using metric exporter node's expression language syntax, with the metric's name inside a selector that filters data based on a separate node identification. Because the query uses a condensed time period (such as the last minute), the provided result accurately reflects the node's current condition. In real-time systems, where obsolete data may result in suboptimal judgments or SLA violations, such a structure enables quick response.

When it executes the query, the algorithm determines whether the output is genuine data. If yes, it fetches the most recent scalar value associated with the query measure and converts it to a standard numeric form so that processing can continue. If the query fails to yield valid data, or if an exception occurs during the query execution, the algorithm fails silently by logging debug data and returning a null-equivalent value. This provides robustness in the event of network loss, momentary Prometheus failure, or node failure.

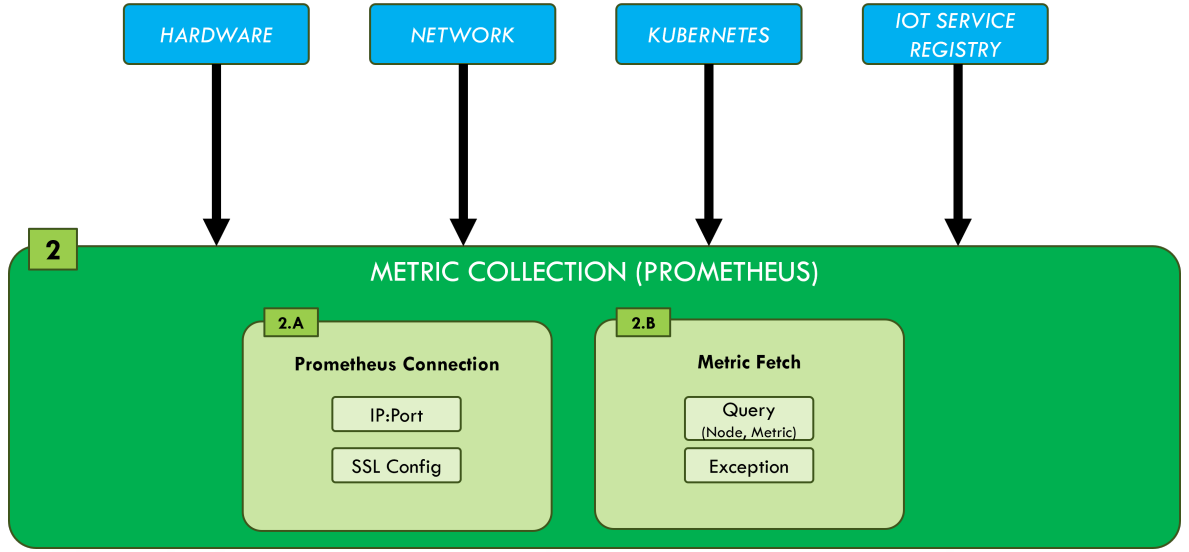


FIGURE 3.7 – Metric Collection &amp; Aggregation Module

### 3.7.1.3 CRIU Integration Module

As explained in the previous sections ; CRIU's and the algorithm's overall goal is two-fold : use a checkpoint file to record the state of a running container's execution, including its memory, process tree, file descriptors, and network information ; and restore the container on a different host from this checkpoint. Proactive and reactive migration strategies are made possible by this end-to-end procedure, which permits a container to be paused on one node and resumed on another node without affecting its execution environment.

Choosing a target container and a directory to hold the checkpoint metadata is the first step in the checkpointing process. To activate the CRIU checkpoint mechanism, the algorithm internally creates and runs a container runtime command. A structured process snapshot directory is created by the command and saved under a specific checkpoint identifier. The function returns success if the action is successful ; if not, it gently handles exceptions and logs the reason for failure.

The destination node's ability to read the checkpoint data and comprehend the CRIU metadata, however, is assumed by the restore logic. In order to start a new container from the checkpoint and continue its execution from the captured point, it first establishes a secure remote session with the destination node and sends a compound command. Container runtime interfaces that are compatible with CRIU can be remotely invoked by centralized orchestration modules to handle container restoration on

any cloud or edge node.

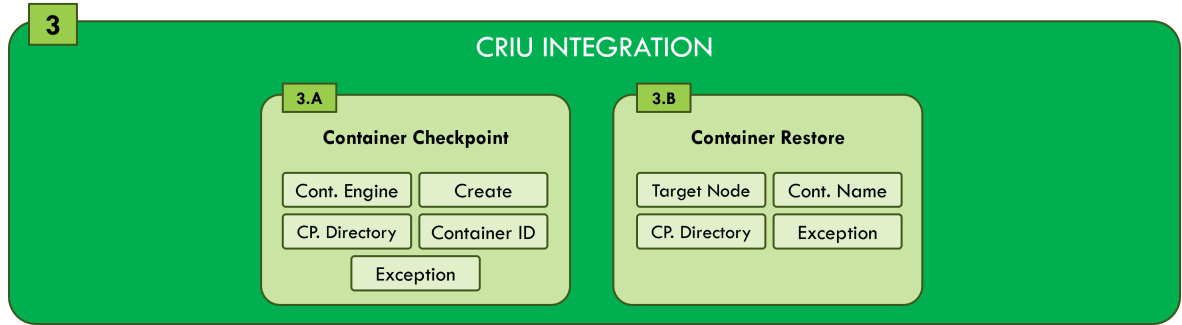


FIGURE 3.8 – CRIU Integration Module

#### 3.7.1.4 Worker Node State Tracking Module

The worker node tracking module is an important component that ensures the system always continuously has a record of the status of each node that is accurate and up to date. First, algorithm creates a array of monitored nodes, where each is identified by its unique identifier. Then, for every node, the system periodically queries resource and performance metrics using the Prometheus-based metrics collection exporters. These include hardware-related data—like CPU and memory usage—network performance indicators, such as latency and bandwidth, and metrics specific to IoT applications, including service priority and mobility.

Over the collection of registered worker nodes, the monitoring loop iterates continually. It retrieves current values for preset metric categories linked to every node by submitting queries to the Prometheus server during each loop. After that, an internal data structure that associates node identifiers with their respective state representations receives these values. Higher-order modules like fuzzy logic engines and migration orchestrators can use the system-wide visibility that this structure provides as a central repository of runtime telemetry.

Instead of using a push-based or event-driven methodology, the system uses a polling-based approach to metric collecting. The practical factors of uniformity, statelessness, and compatibility with Prometheus’s pull-based design served as the foundation for this choice. Additionally, redundancy is avoided and concurrent modules can access the most recent system state without incurring extra data fetching overhead when a centralized, shared data structure is used.

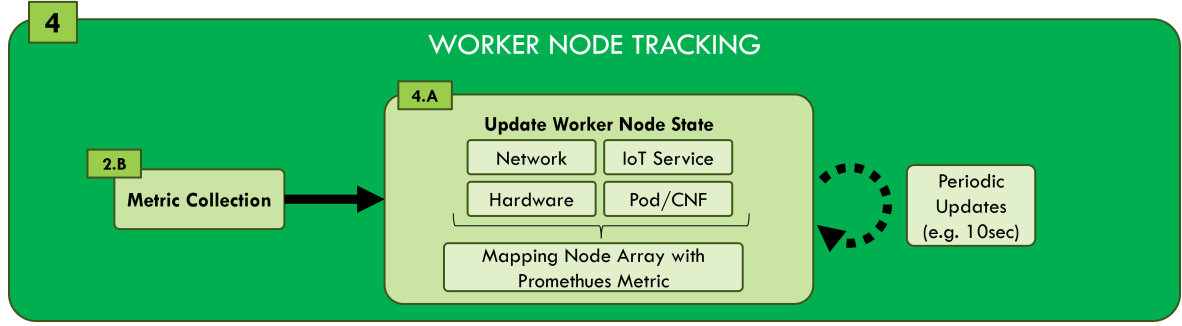


FIGURE 3.9 – Worker Node State Tracking Module

The algorithm is designed to withstand errors in retrieving specific metrics. Only that node's metric value stays stale if a specific metric query fails ; the system keeps updating the others. Even in the event of network outages or partial failures, monitoring will continue thanks to this localized error tolerance.

The module is appropriate for integration into event loops, multi-threaded systems, or reactive orchestration pipelines due to the statelessness of the update function and its deterministic execution interval, which guarantee great predictability.

### 3.7.1.5 Migration Manager Module

Dynamic, context-aware container migration is necessary to maintain service continuity, load balancing, and infrastructure resilience in a distributed edge-cloud system supporting containerized IoT workloads. This module outlines a continuous migration management procedure that combines live migration methodologies, fuzzy decision-making, and real-time monitoring to automatically redistribute workloads. This system exemplifies a crucial element of an autonomic orchestration framework, in which changing runtime conditions dictate container movement rather than statically configured containers.

At the highest level, the module runs an iterative control loop that continuously assesses each worker node's operational state and decides whether to transfer any containerized services. Because the loop runs at set intervals, it is responsive to changes in the environment, including resource saturation, power outages, and service deterioration brought on by mobility. The output of a fuzzy inference engine, which combines several input parameters (such as CPU load, memory pressure, latency, power levels, and mobility state) into a scalar migration urgency score, determines whether to start the

migration for each worker node.

If the fuzzy decision exceeds a certain threshold, the algorithm determines the result to be a signal of service risk or performance degradation and starts the migration procedure. The following are to choose an acceptable target node and conduct a two-phase container relocation : checkpointing in the source node and restoring in the target node.

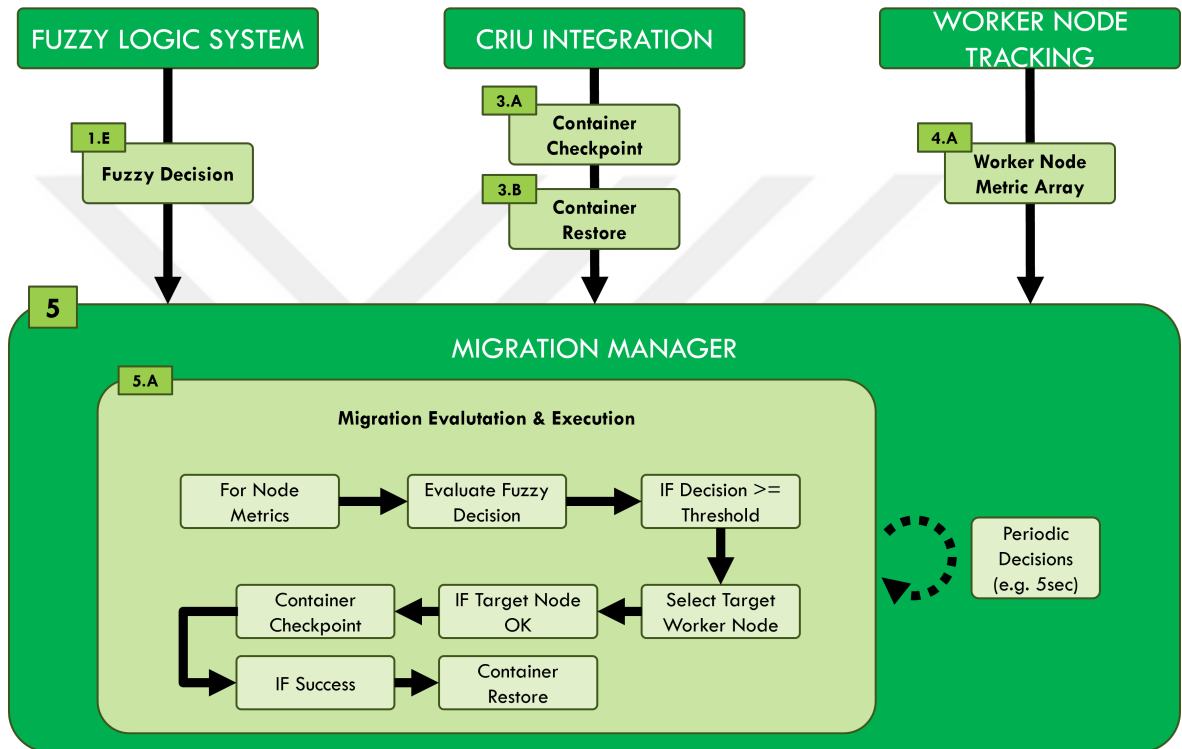


FIGURE 3.10 – Migration Manager Module

Because it necessitates a multi-criteria assessment of every possible candidate node, choosing a migration destination is not simple. The algorithm uses a weighted scoring system that uses real-time performance data to calculate a cost function for every node. In particular, the score penalizes low battery levels, high latency, and excessive CPU and memory usage—all of which indicate a possible bottleneck or constraint in hosting the incoming workload.

The scoring process is also controlled by service-specific priorities. For example, high-priority services may provide additional weight to low-latency or power-stable nodes to satisfy QoS demands. This is accomplished by biasing the composite score toward the nodes more closely aligned with high-priority performance expectations. The target is the candidate node with the lowest score—reflecting most desirable operating

conditions. Importantly, the algorithm ensures the current node is taken out of the candidate set to prevent redundant migrations or self-assignment.

A two-step live migration process is carried out by the system after a suitable migration candidate has been identified. First, the user-space tool (CRIU) that can record a process's entire runtime state is used to checkpoint the container on the source node. Memory, file descriptors, network sockets, and other context are all included in this. A complete snapshot of the container's execution environment is represented by the checkpoint data, which is kept in a temporary directory.

The system then remotely starts a restoration sequence on the destination node, which recreates the container from the serialized state and allows it to resume operation with the least amount of interruption to service. It is appropriate for latency-sensitive and mission-critical Internet of Things applications because it permits a stateful migration that maintains active service sessions.

#### **3.7.1.6 Main Implementation Module**

A modular and distributed container migration system's execution entry point is the module that is being offered. It contains the bootstrap logic in charge of starting and maintaining two ongoing activities at the same time : real-time migration decision-making and execution, and system state monitoring through telemetry aggregation. This orchestration pattern serves as the runtime foundation for the adaptive workload management architecture and incorporates best practices in reactive system design.

## 4 EXPERIMENTS

### 4.1 Test Environment

A mockup simulation environment was created to illustrate the basic ideas of container migration in order to verify the suggested migration approach in advance. The development of fuzzy inference systems of the Mamdani and Sugeno types in a simulation framework based on Python was the main emphasis of this phase. Before being implemented in a real-world testbed, the goal was to assess the fundamental decision-making procedures in a controlled environment and confirm the accuracy and responsiveness of the fuzzy logic-based migration mechanisms.

The data consists of multi-dimensional container, network, and IoT service metrics such as CPU (%) and memory usage (%), network latency (ms), power consumption (W), service priority (1-5), device mobility (0-1) and battery levels (%), randomly generated to simulate a time frame data gathered across various nodes in the infrastructure. The simulation also introduces node failures, i.e., if a node crashes, then its corresponding metrics reach zero, indicating the real unavailability of a node. The fuzzy logic system then determines if there is a need for migration and decides if an immediate or moderate migration action needs to be invoked. Load balancing processes also dynamically adjust target node utilization so that the workloads are optimally distributed among active nodes.

To analyze, the strategy is migration decision analysis and time-series monitoring, whereby the impact of each fuzzy inference system (FIS) is compared. The Mamdani and Sugeno-based decision models are executed concurrently on the identical dataset, and their decisions are recorded per node per timestamp. The system also measures the agreement rate between the two models, or the rate at which they make comparable migration decisions. The accuracy of each model is also established by the degree to which the models concur in their classification of migration urgency. Visualization techniques employed, scatter plots, heatmaps, and time-series migration trends also provide information on the behavior of both models under varying system conditions.

Mamdani and Sugeno FIS logic is same as explained in the previous sections. Experi-

mental results have been demonstrated in the next chapter.

## 4.2 Advanced Simulation

Several system and environmental factors are taken into consideration when evaluating container movement decisions using the fuzzy logic system. The following are the input variables specified in the fuzzy inference system; CPU Usage, Memory Usage, Bandwidth Utilization, Network Latency, Power Consumption, Battery Level, Device Mobility, Service Priority. The migration decision score, which is represented by a single metric and is defined on a continuous scale from 0 to 10, is the output variable.

The `automf()` method is used to automatically assign membership functions (low, medium, and high) to the majority of input variables. However, certain variables are given specially designed trapezoidal and triangular membership functions for improved interpretability and flexibility :

- **Battery Level : Trapezoidal (low, medium, high)**
- **Device Mobility : Triangular (static, mobile)**
- **Service Priority : Triangular (low, high)**
- **Migration Decision (Output) : Triangular (none, moderate, immediate)**

In order to simulate rule assessment utilizing manually determined thresholds for comparative analysis, a crisp logic approximation is also constructed under the Sugeno-style logic.

Human-readable linguistic rules of the following form are used to encode decisions in the system, which has a traditional Mamdani-type rule-based structure : If (A is High) and (B is Poor), then (Decision is Immediate). A representative set of rules including :

- **High CPU and memory usage → Immediate migration**
- **Poor bandwidth and good latency → Moderate migration**
- **Low battery and high power consumption → Immediate migration**
- **Mobile device or high-priority service → Immediate migration**

The rules in the Sugeno variation are constructed manually using conditional logic, simulating the same judgments through crisp mappings based on thresholds.

In real-time, the fuzzy logic evaluator takes current measurements of the system as inputs, processes them via the fuzzy control system, and calculates a migration decision

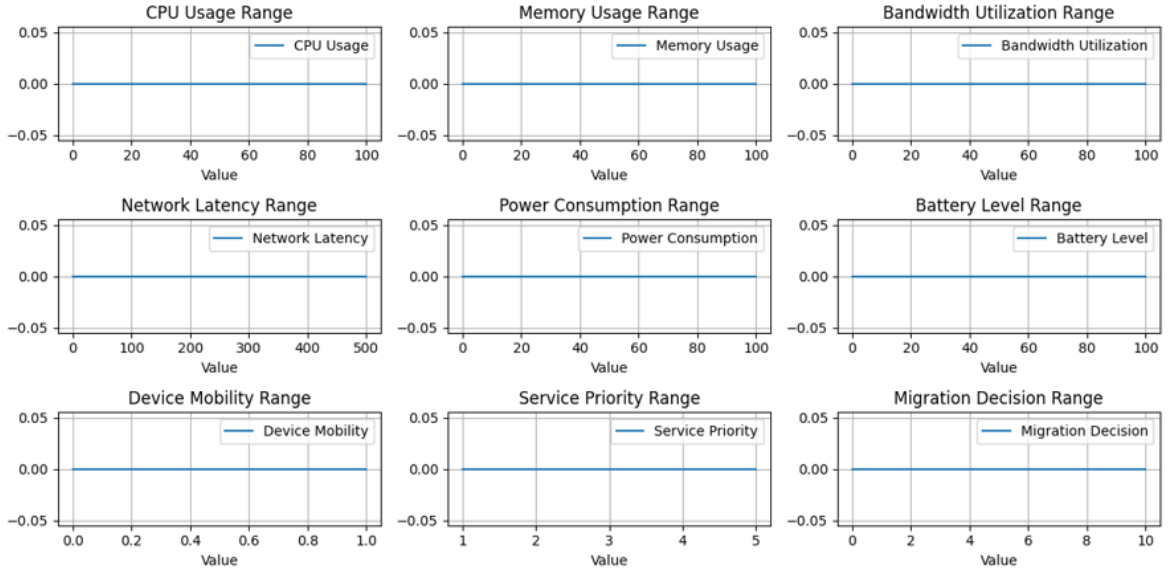


FIGURE 4.1 – Python based Simulation - Input Variables

FIGURE 4.2 – Python based Simulation - Membership Functions

score between 0 and 10. If the score is above a hardcoded threshold of 6, a migration is initiated. Condition-based modeling of migration behavior in changing environments is supported by this modular logic, enabling uniform and adaptive evaluation over simulation iterations.

A collection of plotting functions has been made available to facilitate easier interpretation and analysis of simulation results. These are the visualization of fuzzy decision score distribution, migration events between workers versus time, number of migrations by time versus target worker group, the mean fuzzy decision score per worker, and the migration trigger rate per worker (for scores  $> 6$ ). These graphical results enable evaluation both quantitatively and qualitatively of the migration pattern of the system in altered situations, i.e., during load changes, failure events, and recovery intervals.

### 4.3 Migration Scenarios

In evaluating the effectiveness of the proposed migration framework, four distinct service migration scenarios were identified and intentionally selected for experimental simulation. Each scenario represents a realistic class of events commonly encountered in edge-cloud IoT environments and is designed to demonstrate the decision-making

capabilities and adaptability of the orchestration system.

- **Pod Failure-Driven Migration :** This scenario aims to prevent service disruption resulting from pod-level failures, logic errors, or Kubernetes-related anomalies. Upon detecting such faults, the system initiates the migration of the affected pod to the most suitable available worker node, thereby ensuring service continuity.
- **Resource Optimization-Based Migration :** In this case, system-level resource metrics—such as CPU usage, memory availability, and network bandwidth—are continuously monitored to proactively assess the need for migration. When suboptimal performance is detected, the orchestrator selects a target node based on overall resource availability and operational health to facilitate balanced workload distribution.
- **New Worker Node Addition-Driven Migration :** This scenario, closely related to resource optimization, is triggered by the deployment of a new worker node within the cluster. Beyond deploying new services on the added node, the orchestrator also redistributes existing pods from overloaded nodes, thereby enhancing overall load balancing and optimizing cluster performance.

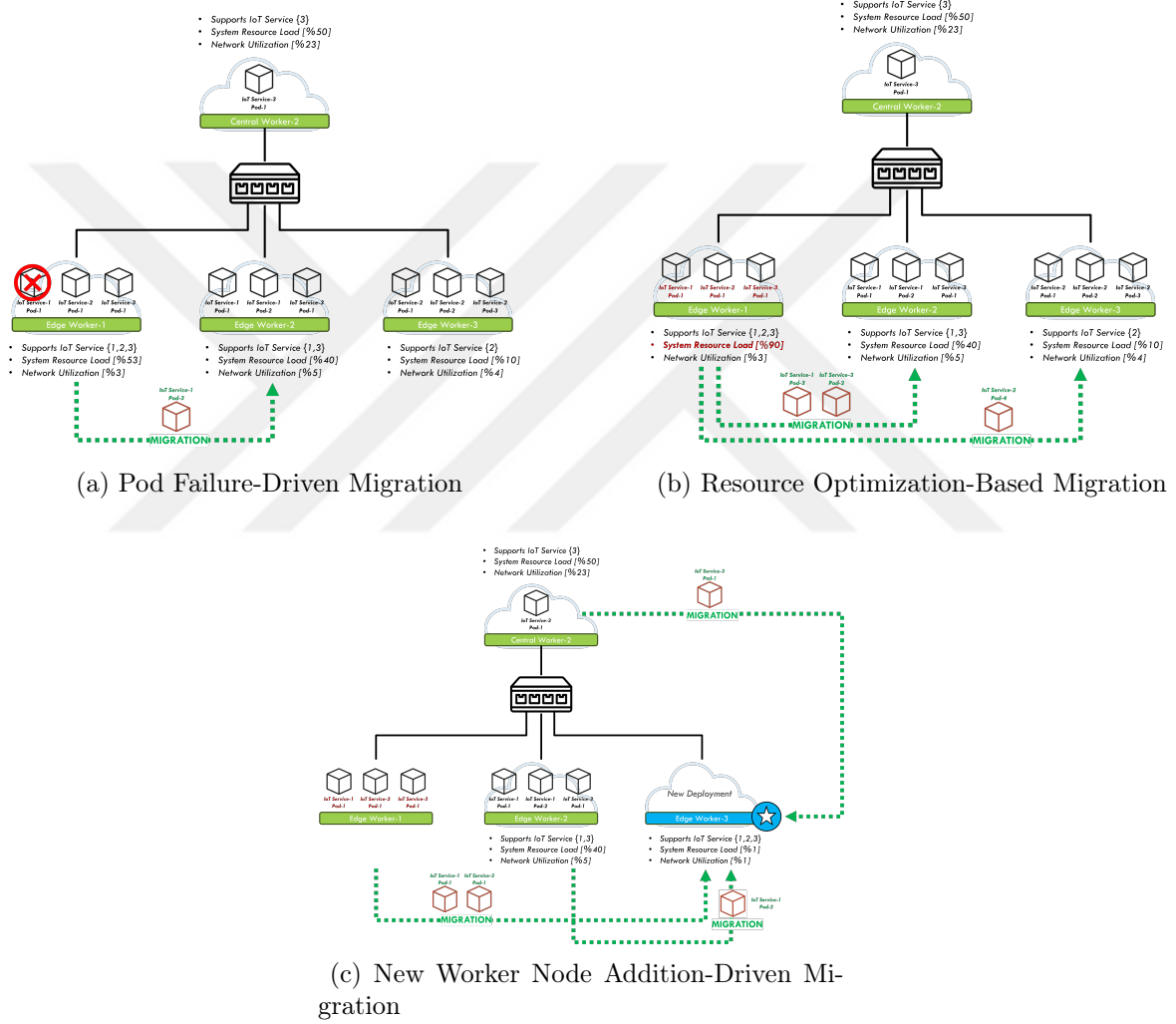


FIGURE 4.3 – Migration Scenarios

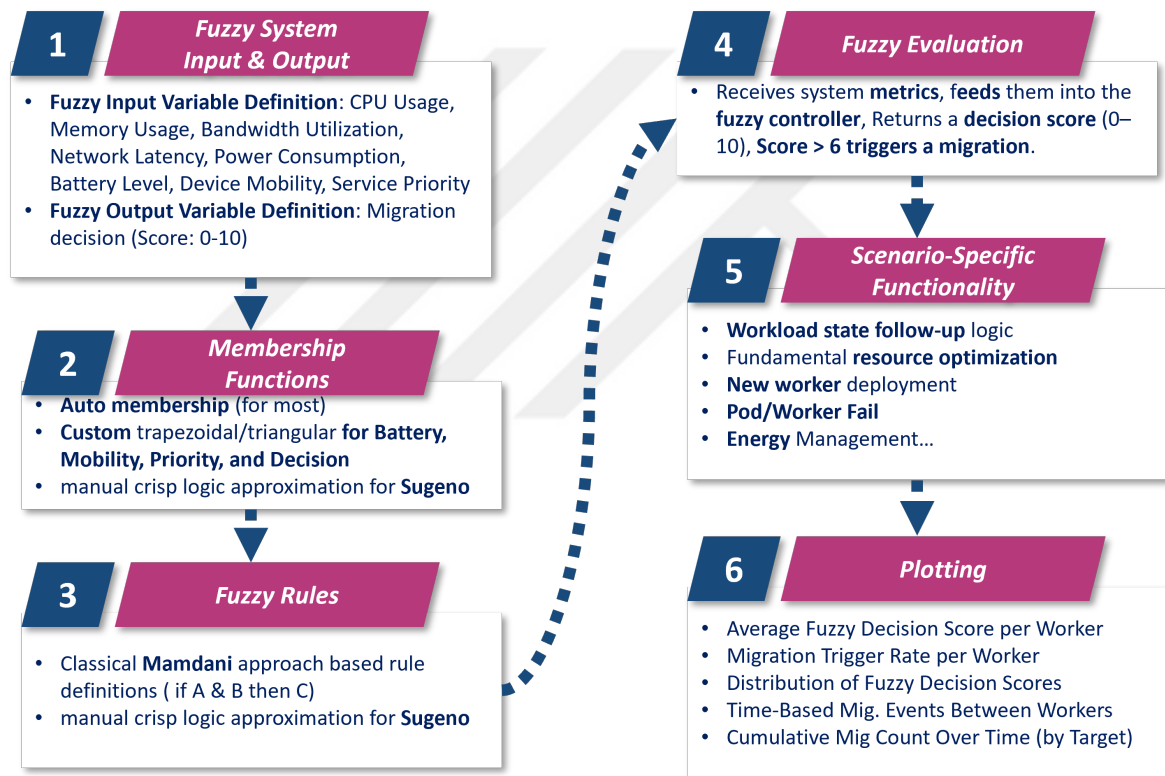


FIGURE 4.4 – Python based Simulation Logic

## 5 DISCUSSION & CONCLUSION

The experimental verification is outlined in two stages. During the first stage, a mockup simulation is established to demonstrate the basic migration logic. During this stage, the focus is on implementing both Mamdani- and Sugeno-type fuzzy inference systems using a Python-based simulation environment to validate the underlying decision-making processes. The second phase entails the design of an enhanced simulation implementation to realize modular system design in basic sense, while also exploring variety of container migration scenarios using a controlled randomly generated data.

### 5.1 Mamdani & Sugeno-based Migration Simulation based Discussions

#### 5.1.1 Basic Simulation

In first implementation studies, very fundamental logic has been established to simulate the system. Randomized data has been utilized for demonstration purposes. In this sense, The following plots visually represent and summarize the fuzzy rules and membership functions employed in the Mamdani- and Sugeno-based decision-making processes.

To give an illustration of the distinction, it can be used for the example case where a

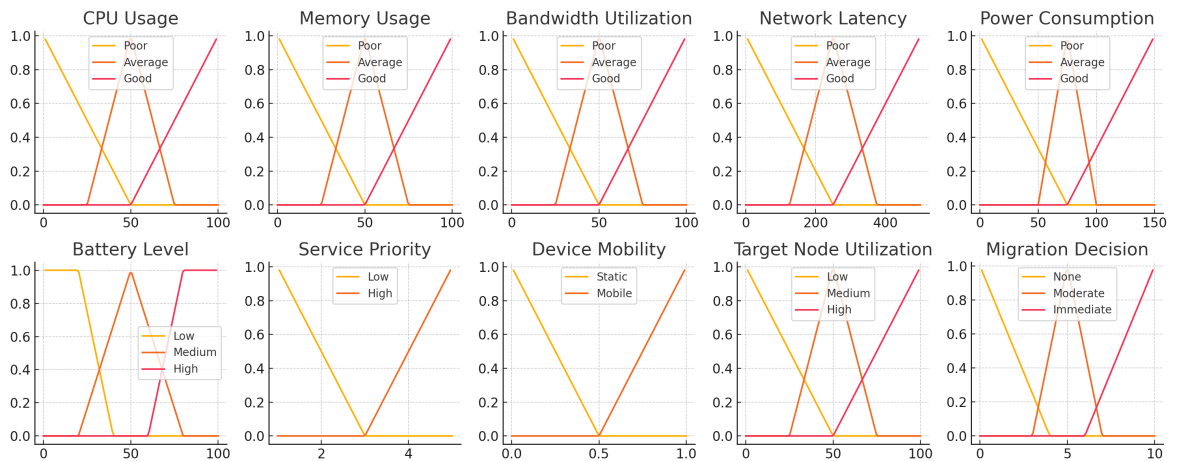


FIGURE 5.1 – Fuzzy Rules & Membership Functions

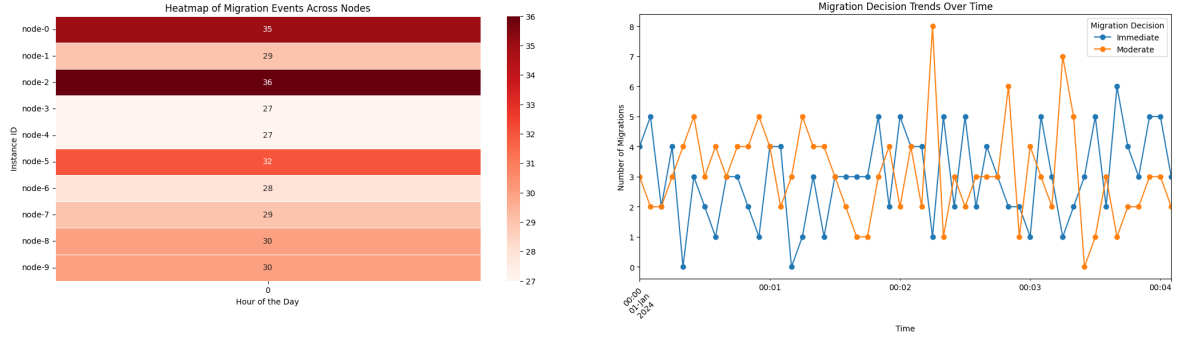


FIGURE 5.2 – Heatmap of Migration Events Across Nodes & Migration Decision Over Time Trend

container on **Node-3** has **CPU utilization of 85%**, **memory utilization of 70%**, and **network latency of 150ms**. In the Mamdani system, the fuzzy rules classify these inputs as *high CPU utilization*, *high memory utilization*, and *moderate latency*, which are combined into an aggregated fuzzy output, defuzzified to a crisp migration decision of :

$$D_{\text{Mamdani}} = 7.2 \quad (\text{moderate to high migration urgency}) \quad (5.1)$$

In contrast, the Sugeno model, using weighted average calculations, might compute a direct migration urgency score of :

$$D_{\text{Sugeno}} = 6.8 \quad (5.2)$$

reflecting a **slightly different but computationally optimized** decision.

The level-constrained simulations demonstrate that fuzzy logic-based paradigms can be appropriately configured for IoT-like, complex, multi-level network structures for optimizing infrastructure-level migration choices as well as in general resource management.

### 5.1.2 Advanced Simulation

Three test cases has been executed to take a closer look at how the fuzzy-based migration system behaves and performs : (1) New Worker Deployment, (2) Pod/Worker Failure, and (3) a comparison between Mamdani and Sugeno Fuzzy Logic.

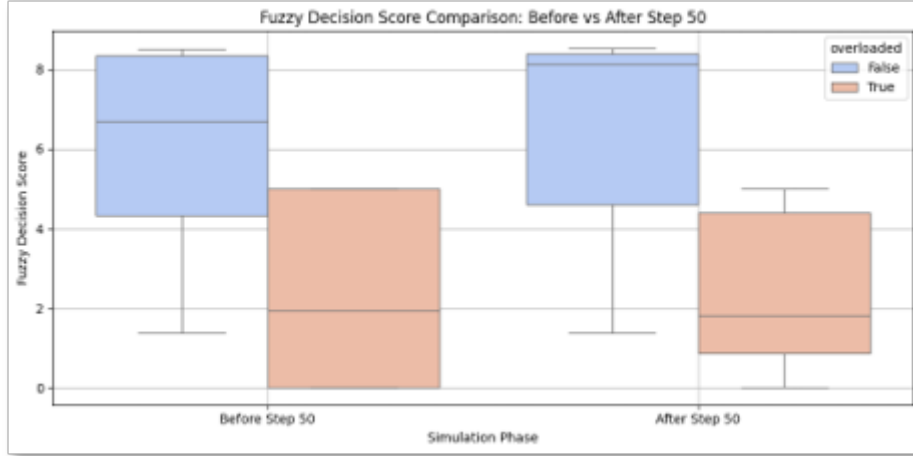


FIGURE 5.3 – New Worker Deployment - Box Plot

In the scenario where new worker has been added, the system runs for 100 simulation time-steps. During this time, system metrics are generated randomly for each worker at every time-step, thanks to our state-tracking logic. At time-step 50, a new node has been introduced, worker5. The simulation utilizes batch execution framework, where fuzzy decisions are made at each step using a Mamdani-based fuzzy inference system. Migrations implemented when the fuzzy decision measure goes above 6. The simulation logs keep a detailed record of every migration event, noting the source and target worker, the type of workload, and the migration cost. The cost varies by workload type : IoT workloads are assigned a cost of 1, video workloads a cost of 3, and AI workloads a cost of 5. Additionally, the system identifies overloaded workers based on a mix of CPU usage, memory usage, bandwidth usage, and network latency. To help with monitoring over time, the simulation code has a feature for tracking and caching data, which allows to study how the system state evolves to minimize randomness. One achieved observation from the results is that after introducing worker5 at step 50, redistribution of the load can be seen on every simulation execution. This is backed up by a shift in migration targets toward the new worker node and a noticeable drop in the number of workers flagged as overloaded in the following steps.

In scope of pod/worker failover scenario, the simulation runs for 100 time steps to process worker system parameters using a state-tracking method. Failure is introduced at time-step 50 which Worker4 has the failure-like behavior, i.e., show low performance metrics. The Mamdani-based fuzzy logic method is utilized to determine system conditions and the consequent migration actions. Migrations are initiated if the fuzzy decision score surpasses a threshold of 6. Workers are flagged as overloaded based on

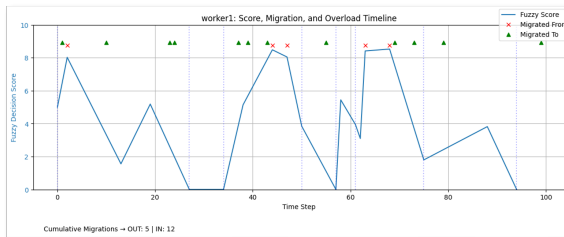
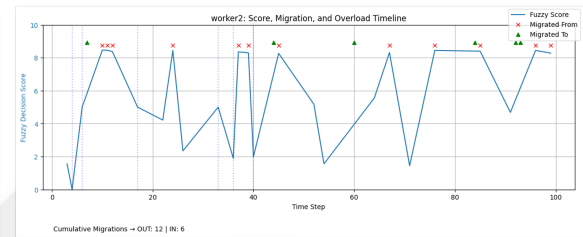
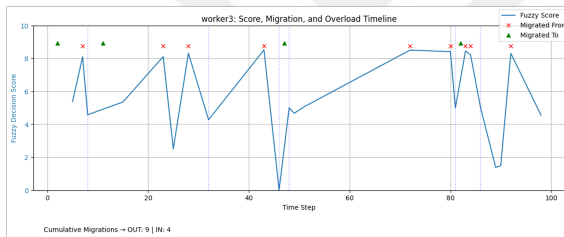
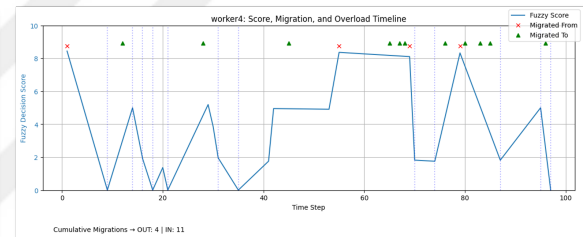
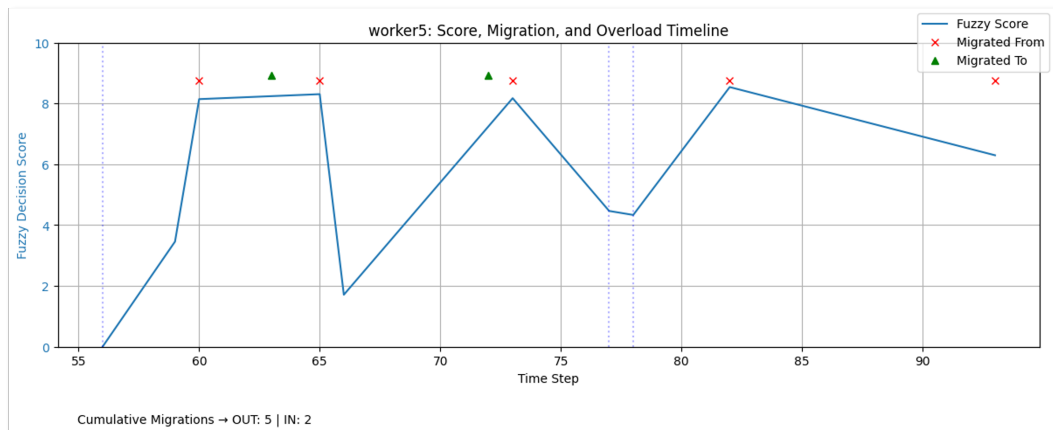
**Worker-1****Worker-2****Worker-3****Worker-4****Worker-5**

FIGURE 5.4 – New Worker Deployment - Migration Plots

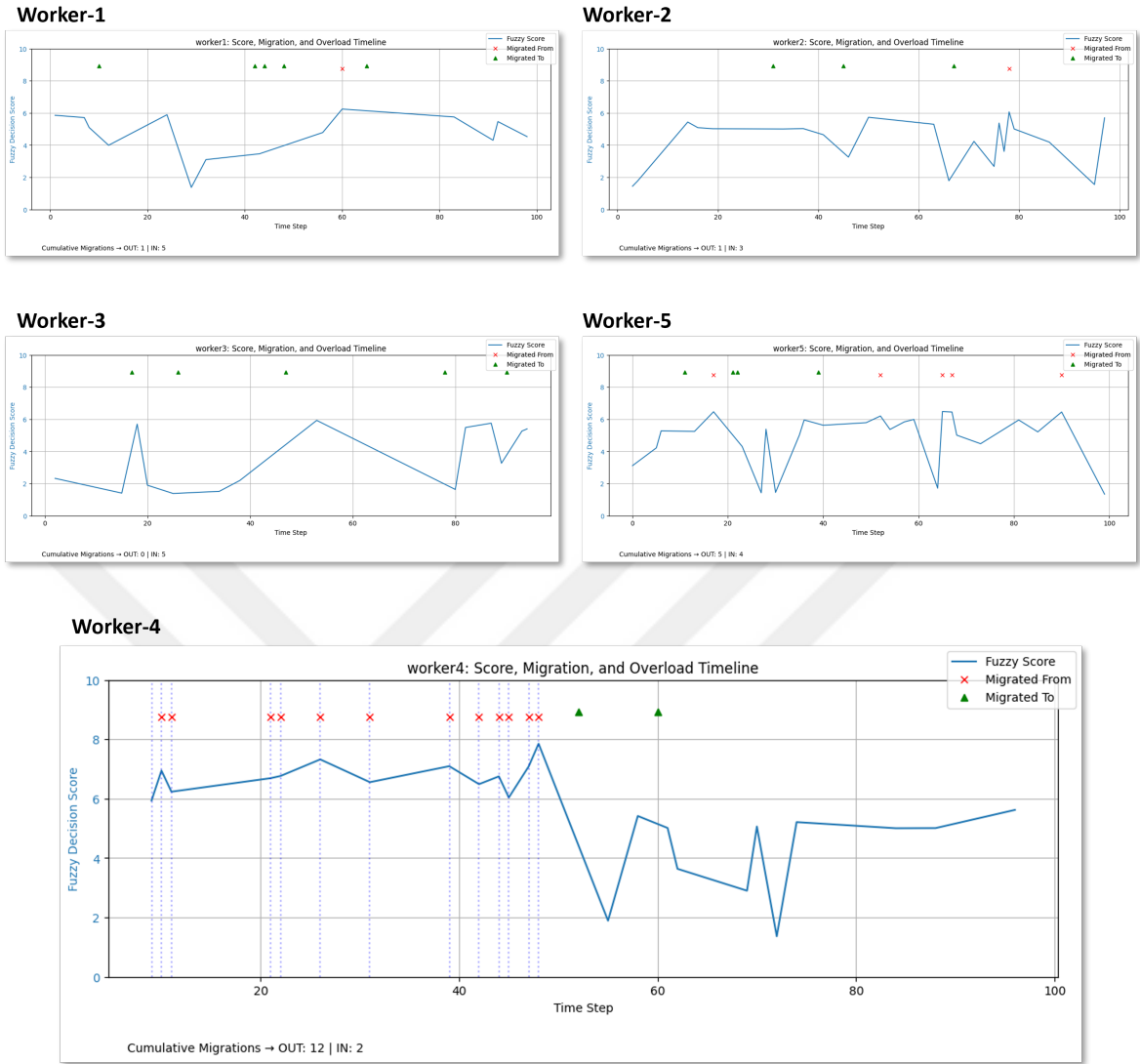


FIGURE 5.5 – Worker Failover - Migration Plots

the of CPU, memory, bandwidth utilization and network latency parameters. After the step 50, other worker nodes began gather significant rate of the worker-4 load. This demonstrates the applicability of the logic as workloads being reallocated by the migration logic due to node failure. In addition, worker-4 was also the most migration triggered worker, which means that the fuzzy decision-making works as expected.

In the Mamdani vs. Sugeno experiment, two systems has been compared again in 100 time-steps using randomly created but tracked data. Both Mamdani and Sugeno fuzzy systems were used separately to see how they acted in the same conditions. Migration decisions were gathered based on the same rule : a migration happens if the fuzzy score is higher than 6. For every migration, various data for enrichment has been stored, including the type of workload, the source and target worker pair, and the cost of migration.

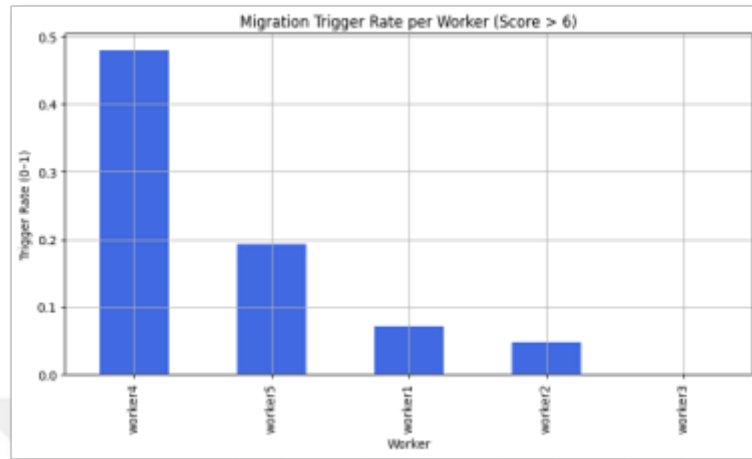


FIGURE 5.6 – Worker Failover - Migration Rate Plot

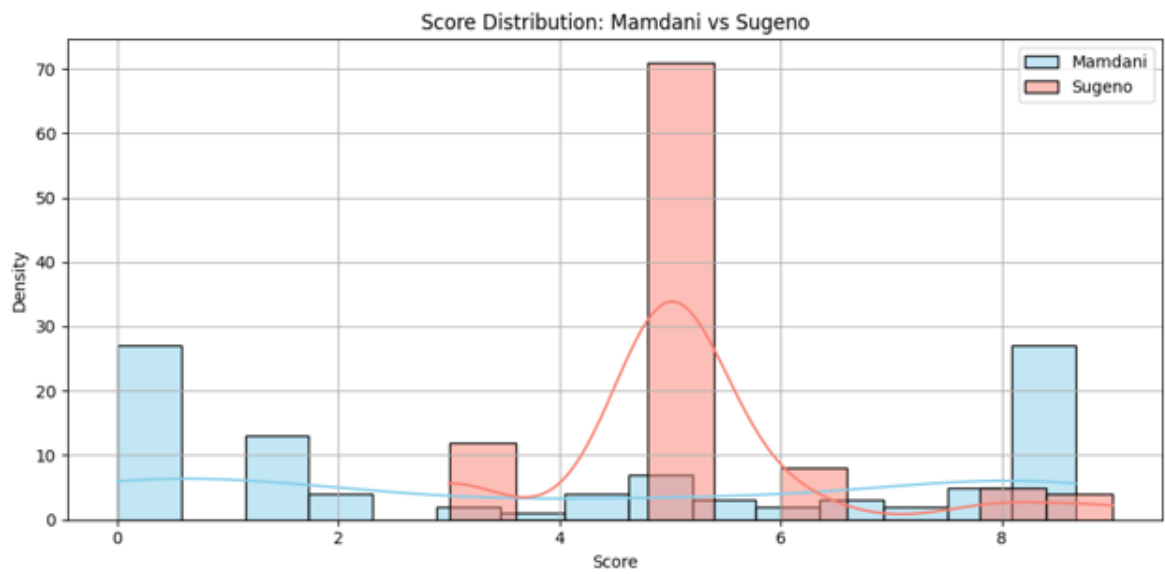


FIGURE 5.7 – Mamdani vs. Sugeno - Score Distribution

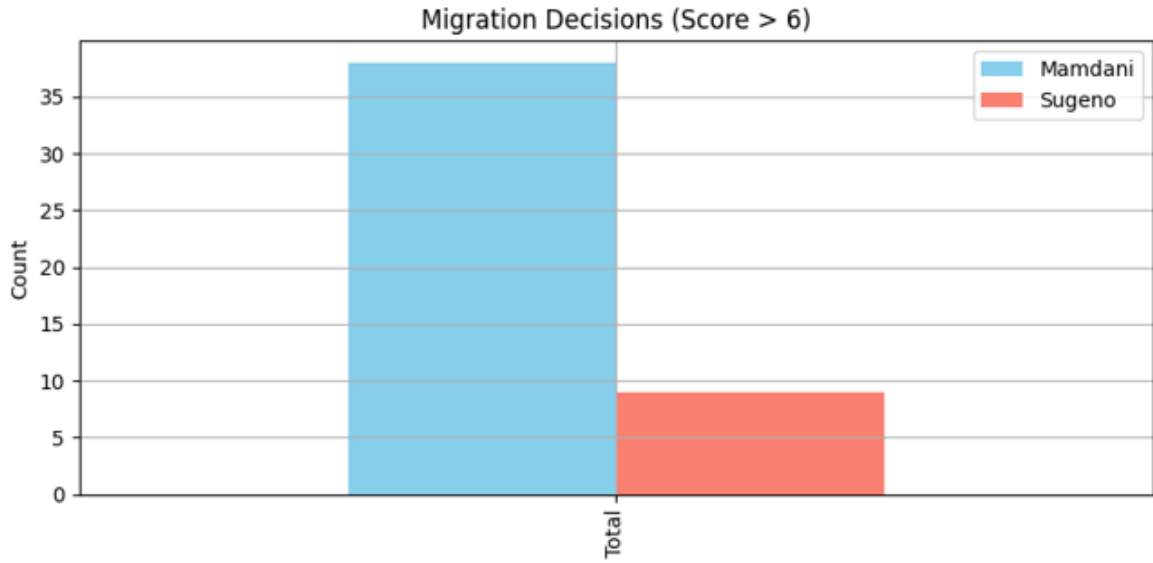


FIGURE 5.8 – Mamdani vs. Sugeno - Total Migration Decisions

These results indicated clear differences in behavior in the two fuzzy methodologies. Sugeno-based decisions tended to cluster closely around a score of around 5, with a more steady and cautious style of decision-making. On the other hand, there was a larger spread of scores in the Mamdani-based system, with large numbers of values around 0 and 9, with more reaction and fluctuation in response. The implications of these results are that Sugeno inference is more stable.

## 5.2 Conclusion

As elaborated in the previous chapters, this research intends to solve two main issues ; the creation of an infrastructure that transcends conventional centralized computing models through dynamic service management and optimization at the edge, and the development of effective decision-making and orchestration strategies specific to various, service-based IoT environments. For this purpose, the proposed framework leverages a low-cost, lightweight infrastructure comprised of edge-cloud nodes and a minimal version of Kubernetes distribution combined with a fuzzy-logic-based decision-making system and a modular orchestration system.

Unlike previous studies, which put emphasis on single-level metrics or selective optimization targets, this work is more end-to-end and integrated in approach—ranging from hardware-layer telemetry to IoT service-layer metrics. The proposed architecture emphasizes service-oriented governance to facilitate centralized monitoring and centra-

lized control with distributed, edge-level processing still being achieved. Furthermore, by providing fuzzy logic as a light-weight, interpretable surrogate for computationally costly optimization or machine learning, the paper demonstrates the low-cost real-time decision-making feasibility in containerized environments. This is particularly pertinent in application scenarios where aggregation of heterogeneous metrics is needed for dynamic orchestration of workloads, an area that remains under-explored in current edge computing research.

Although empirical evaluation results specific to service migration in realistic deployment environments are not available, this paper offers a systematic foundation for the performance analysis of container-based service migration in various scenarios and algorithmic solutions, addition to the proposing an architectural perspective for IoT edge-cloud resource management. The evaluation objectives have been developed in alignment with the major architectural motivations of the project and are designed to cover both system-level and service-level implications of dynamic migration in edge-cloud environments.

In particular, three migration scenarios—namely pod failure-driven, resource optimization-driven and new worker deployment-driven—have been identified as realistic environments where intelligent orchestration is expected to enhance system efficiency and resilience. The proposed evaluation framework includes the comparative assessment of Mamdani and Sugeno fuzzy inference systems as decision-making engines in terms of their operational behavior in these scenarios.

Although current results are limited to system design and simulation-level validation, this evaluation plan is the basis for future experimental campaigns. It also positions the proposed architecture as a lightweight and interpretable solution in contrast to more complex migration strategies, particularly in resource-constrained IoT infrastructures where computational efficiency, service continuity, and energy awareness are of primitive concern.

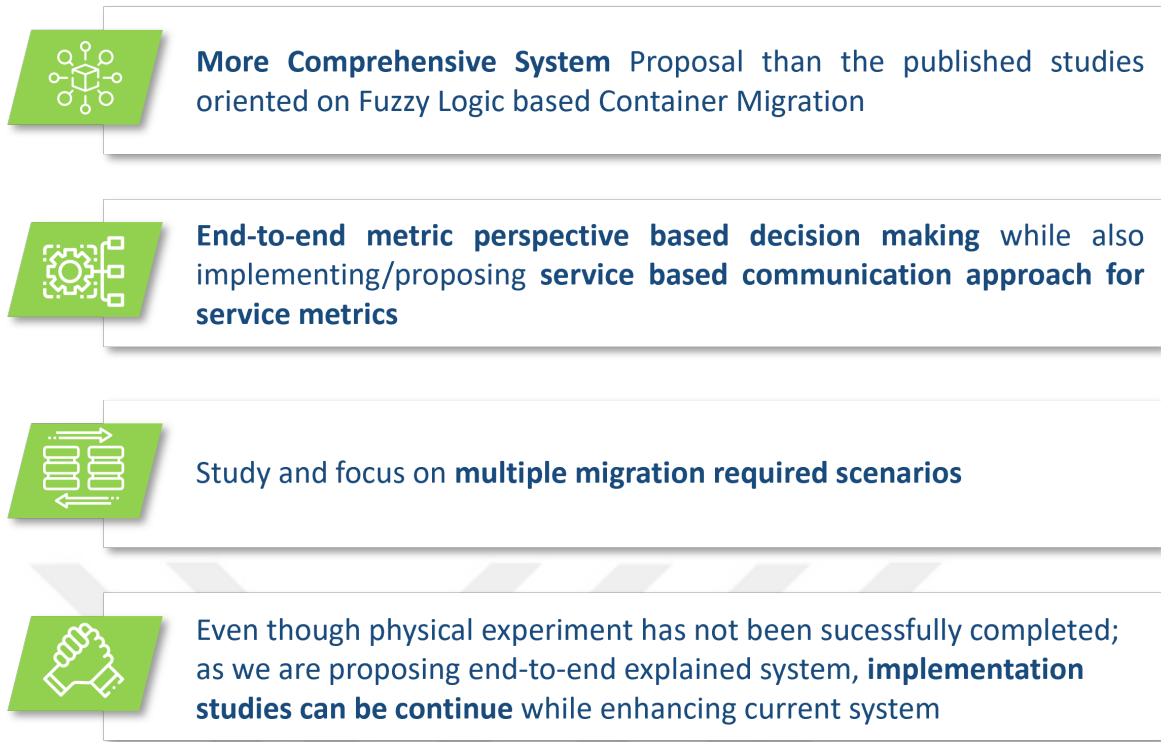


FIGURE 5.9 – Offers of the proposed approach

### 5.3 Future Studies

As there are a number of possible directions for further research to improve and validate this suggested IoT container migration framework. In order to capture a wider range of real-world dynamics and validate the proposed approaches advantages and disadvantages; real life simulation requires to be implemented. There have been efforts to construct a edge-cloud IoT network & migration orchestration framework and the following details in the next sub section can be utilized as reference for future real-life environment study.

Besides the implementation, the next step is to integrate and test new migration scenarios that include energy-aware migration, device mobility, schedule-driven maintenance, and periodic load balancing. It would be possible to construct large-scale deployment evaluation and inter-cluster migration behavior on a more realistic basis if the existing framework were extended to accommodate multi-cluster designs, both in simulated settings and on real hardware. Extending the migration system to general-purpose live migration approach, improving the management of different kinds of containers is another significant area of work. Lastly, expanding on a hybrid decision-making method that combines a fuzzy logic scheme with a backup system based on optimization or

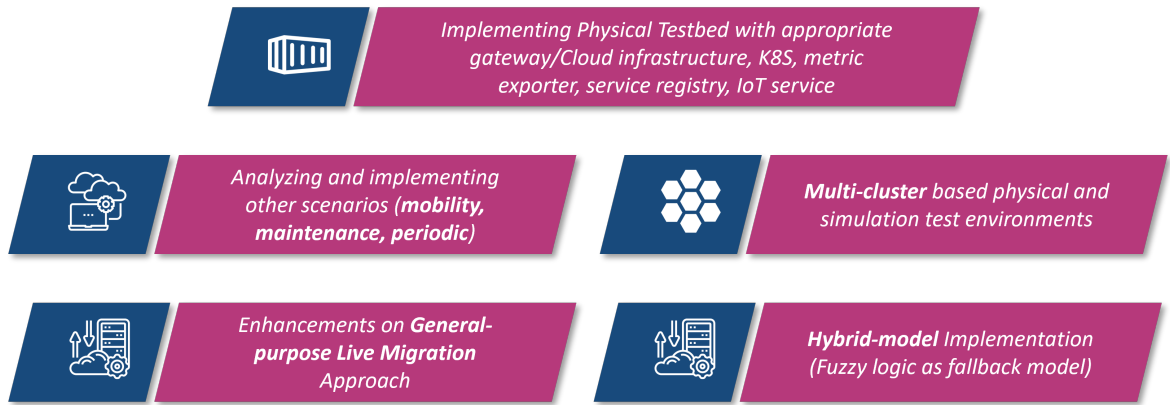


FIGURE 5.10 – Future study

machine learning techniques might both stabilize and adapt to changing operational conditions. In general, these initiatives seek to improve the system’s adaptability, scalability, and robustness to facilitate deployment in intricate and ever-changing IoT edge-cloud environments.

### 5.3.1 Physical Test Environment Study

As the proposed container-based IoT project is designed to prove a new distributed workload migration system with lightweight infrastructure, experiments can be conducted with various physical demonstration methodologies. In the physical test environment study attempts; The centralized cloud, hosted on a Raspberry Pi 3 Model B+, acts as the K3S master and a worker node, allowing centralized IoT workloads. This environment includes an IoT service registry, powered by Consul, in order to maintain an updated record of the services and configurations of the IoT end devices. This registry is important for monitoring service-level metrics in IoT systems, including parameters such as service availability, error rates, and TPS. The centralized cloud system interacts fluidly with edge gateways through a basic Ethernet switch, resulting in an integrated, yet modular network topology.

The edge infrastructure consists of Raspberry Pi 3 devices that work as worker nodes and deploy IoT workloads in containerized pods. The gateways can be interfaced with wireless IoT sensors and simulated devices emulating interactions with real end devices. In addition, to make live migration of workloads possible across different edge nodes or to the central cloud, a container live migration solution like CRIU is leveraged for the horizontal and vertical migrations of the former. Prometheus has been deployed

for comprehensive metric collection, collecting hardware-level data (CPU, memory, power consumption), network-level metrics such as bandwidth, latency, and packet loss, container-level performance indicators including resource utilization and checkpoint error rates, and IoT service-level parameters such as service priority and availability. This planned to feed into a Python-based fuzzy logic decision-making model that will evaluate these metrics in real time to orchestrate optimized migration scenarios such as failover recovery, energy efficiency, workload optimization, and mobility support.

As mentioned, the basic structure of communication is provided by an ethernet switch, which provides stable and low-latency Ethernet connections between the cloud and edge nodes. The setup emulates a real network environment, hence providing a controlled testing platform for the analysis of migration decisions and their impact on system performance. The introduction of fuzzy logic enhances the decision process by allowing the simultaneous consideration of multi-layered metrics in all nodes, hence facilitating dynamic and adaptive workload balancing. This approach not only shows the feasibility of intelligent migration strategies, but also exposes the scalability and resilience of container-based IoT architectures in diverse operational scenarios.

While deploying services (pods) on the kubernetes it is prominent to build a reliable cluster while oversee network, storage, service configuration, deployment requirements and dependencies. In this sense, It will be much efficient to have a system services work as intended while data collection and extraction are organized and defectless. It is also significant to mention that IoT service traffic needs to be complex while also systematic according to the scenario. To provide this, custom-created containers which includes systematic IoT service data/process can be deploy on edge and cloud machines.

Moreover, Key performance indicators important to evaluate the migration related processes. And migration solution requires to be studied under different FIS algorithms, migration scenarios with different sub-circumstances for comparison and evaluation of the system under qualified KPIs. Following metrics can be explored :

- **total migration time (which captures decision-making latency, checkpoint creation, and restore time)**
- **average service downtime**
- **system response time during migration**
- **success and failure rates of migration attempts**
- **jitter experienced during live migration**

— **network overhead measures**

These metrics need to offer a comprehensive basis for both intra-architectural comparison and cross-study comparison against other methods.



## REFERENCES

- Abdullah, D. B., Hadeed, W. et al. (2022). Container live migration in edge computing : a real-time performance amelioration, *International Journal of Applied Science and Engineering* **19**(3) : 1–8.
- Aguzzi, C., Gigli, L., Sciallo, L., Trotta, A. and Di Felice, M. (2020). From cloud to edge : Seamless software migration at the era of the web of things, *IEEE Access* **8** : 228118–228135.
- Alam, M., Rufino, J., Ferreira, J., Ahmed, S. H., Shah, N. and Chen, Y. (2018). Orchestration of microservices for iot using docker and edge computing, *IEEE Communications Magazine* **56**(9) : 118–123.
- Aleyadeh, S., Moubayed, A., Heidari, P. and Shami, A. (2022). Optimal container migration/re-instantiation in hybrid computing environments, *IEEE Open Journal of the Communications Society* **3** : 15–30.
- Barbalace, A., Karaoui, M. L., Wang, W., Xing, T., Olivier, P. and Ravindran, B. (2020). Edge computing : The case for heterogeneous-isa container migration, *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp. 73–87.
- Bulej, L., Bureš, T., Filandr, A., Hnětynka, P., Hnětynková, I., Pacovský, J., Sandor, G. and Gerostathopoulos, I. (2021). Managing latency in edge–cloud environment, *Journal of systems and software* **172** : 110872.
- Campanile, L., Iacono, M., Marulli, F., Mastroianni, M. and Mazzocca, N. (2020). Toward a fuzzy-based approach for computational load offloading of iot devices, *Journal of Universal Computer Science* **26** : 1455–1474.
- Chen, S. and Zhou, M. (2021). Evolving container to unikernel for edge computing and applications in process industry, *Processes* **9**(2) : 351.
- Chen, Y., He, S., Jin, X., Wang, Z., Wang, F. and Chen, L. (2023). Resource utilization and cost optimization oriented container placement for edge computing in industrial internet, *The Journal of Supercomputing* **79**(4) : 3821–3849.

- Chen, Y., Sun, Y., Wang, C. and Taleb, T. (2022). Dynamic task allocation and service migration in edge-cloud iot system based on deep reinforcement learning, *IEEE Internet of Things Journal* **9**(18) : 16742–16757.
- Das, R. and Sidhanta, S. (2023). Live migration of containers in the edge, *SN Computer Science* **4**(5) : 479.
- Dupont, C., Giaffreda, R. and Capra, L. (2017). Edge computing in iot context : Horizontal and vertical linux container migration, *2017 Global Internet of Things Summit (GIIoTS)*, IEEE, pp. 1–4.
- Fard, H. M., Prodan, R. and Wolf, F. (2020). A container-driven approach for resource provisioning in edge-fog cloud, *Algorithmic Aspects of Cloud Computing : 5th International Symposium, ALGO CLOUD 2019, Munich, Germany, September 10, 2019, Revised Selected Papers 5*, Springer, pp. 59–76.
- Gand, F., Fronza, I., El Ioini, N., Barzegar, H. R., Azimi, S. and Pahl, C. (2021). Fuzzy container orchestration for self-adaptive edge architectures, *Proceedings of a relevant conference (please insert)*.
- He, T., Toosi, A. N. and Buyya, R. (2023). Efficient large-scale multiple migration planning and scheduling in sdn-enabled edge computing, *IEEE Transactions on Mobile Computing* .
- Kim, S.-H. and Kim, T. (2023). Local scheduling in kubeedge-based edge computing environment, *Sensors* **23**(3) : 1522.
- Liu, Y., Lan, D., Pang, Z., Karlsson, M. and Gong, S. (2021). Performance evaluation of containerization in edge-cloud computing stacks for industrial applications : A client perspective, *IEEE Open Journal of the Industrial Electronics Society* **2** : 153–168.
- Lou, J., Luo, H., Tang, Z., Jia, W. and Zhao, W. (2022). Efficient container assignment and layer sequencing in edge computing, *IEEE Transactions on Services Computing* **16**(2) : 1118–1131.
- Ma, Z., Shao, S., Guo, S., Wang, Z., Qi, F. and Xiong, A. (2020). Container migration mechanism for load balancing in edge network under power internet of things, *IEEE Access* **8** : 118405–118416.
- Marchese, A. and Tomarchio, O. (2022). Network-aware container placement in cloud-edge kubernetes clusters, *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, IEEE, pp. 859–865.

- Mutichiro, B., Tran, M.-N. and Kim, Y.-H. (2021). Qos-based service-time scheduling in the iot-edge cloud, *Sensors* **21**(17) : 5797.
- Nakanishi, K., Suzuki, F., Ohzahata, S., Yamamoto, R. and Kato, T. (2020). A container-based content delivery method for edge cloud over wide area network, *2020 International Conference on Information Networking (ICOIN)*, IEEE, pp. 568–573.
- Oleghe, O. (2021). Container placement and migration in edge computing : Concept and scheduling models, *IEEE Access* **9** : 68028–68043.
- Rausch, T., Rashed, A. and Dustdar, S. (2021). Optimized container scheduling for data-intensive serverless edge computing, *Future Generation Computer Systems* **114** : 259–271.
- Ray, K., Banerjee, A. and Narendra, N. C. (2020). Proactive microservice placement and migration for mobile edge computing, *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, IEEE, pp. 28–41.
- Rezazadeh, A., Abednezhad, D. and Lutfiyya, H. (2022). Migror : Mitigating downtime in mobile edge computing, an extension to live migration, *Procedia Computer Science* **203** : 41–50.
- Scolati, R., Fronza, I., El Ioini, N., Samir, A., Barzegar, H. R. and Pahl, C. (2020). A containerized edge cloud architecture for data stream processing, *Cloud Computing and Services Science : 9th International Conference, CLOSER 2019, Heraklion, Crete, Greece, May 2–4, 2019, Revised Selected Papers 9*, Springer, pp. 150–176.
- Şenel, B. (2023). *Container Orchestration for the Edge Cloud*, PhD thesis, Sorbonne Université.
- Shao, S., Tian, S., Guo, S. and Qiu, X. (2023). Container-based internet of vehicles edge application migration mechanism., *Computers, Materials & Continua* **75**(3).
- Singh, A., Aujla, G. S. and Bali, R. S. (2021). Container-based load balancing for energy efficiency in software-defined edge computing environment, *Sustainable Computing : Informatics and Systems* **30** : 100463.
- Singh, G. and Singh, P. (2021). A taxonomy and survey on container migration techniques in cloud computing, *Sustainable Development Through Engineering Innovations : Select Proceedings of SDEI 2020* pp. 419–429.

- Svorobej, S., Bendeache, M., Griesinger, F. and Domaschka, J. (2020). Orchestration from the cloud to the edge, *The Cloud-to-Thing Continuum : Opportunities and Challenges in Cloud, Fog and Edge Computing* pp. 61–77.
- Tang, B., Luo, J., Obaidat, M. S. and Vijayakumar, P. (2023). Container-based task scheduling in cloud-edge collaborative environment using priority-aware greedy strategy, *Cluster Computing* **26**(6) : 3689–3705.
- Tang, Z., Mou, F., Lou, J., Jia, W., Wu, Y. and Zhao, W. (2023). Multi-user layer-aware online container migration in edge-assisted vehicular networks, *IEEE/ACM Transactions on Networking* .
- Xing, T., Barbalace, A., Olivier, P., Karaoui, M. L., Wang, W. and Ravindran, B. (2022). H-container : Enabling heterogeneous-isa container migration in edge computing, *ACM Transactions on Computer Systems (TOCS)* **39**(1-4) : 1–36.

## Appendix A PUBLICATIONS

The following publication has arisen from the work presented in this thesis:

1. M. B. Pala and Ö. Pinarer, “Container Migration Approach for IoT-Based Edge-Cloud Systems,” in *Intelligent and Fuzzy Systems (INFUS 2025)*, Lecture Notes in Networks and Systems, vol. 1531, C. Kahraman et al. (eds.), Springer, Cham, 2025, pp. 824–831. DOI: 10.1007/978-3-031-98304-7\_89.