REINFORCEMENT LEARNING MICROSERVICES SCHEDULER IN INTELLIGENT EDGE COMPUTING

by

Kevin E. Afachao

Submitted in partial fulfillment of the requirements for the degree
Master of Engineering (Computer Engineering)

in the

Department of Electrical, Electronic and Computer Engineering
Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

July 2024

© University of Pretoria

SUMMARY

REINFORCEMENT LEARNING MICROSERVICES SCHEDULER IN INTELLIGENT EDGE COMPUTING

by

Kevin E. Afachao

Supervisor:	Prof. Adnan M. Abu-Mahfouz
Co-supervisor:	Prof. Gerhard P. Hancke
Department:	Electrical, Electronic and Computer Engineering
University:	University of Pretoria
Degree:	Master of Engineering (Computer Engineering)
Keywords:	Artificial intelligence, Cloud computing, Edge computing, Industry, Innovation and infrastructure (SDG 9), Internet of things, Microservices, Reinforcement learning, Resource management, Scheduling algorithms.

The proliferation of internet of things (IoT) devices and resource-intensive applications has necessitated the development of intelligent edge computing frameworks. These frameworks aim to address challenges in the resource management, service latency, and data privacy of IoT devices. This research investigates the complex problem of microservice scheduling within intelligent edge computing environments. The focus is on optimising quality of service (QoS) metrics such as the latency, network bandwidth utilisation, and energy consumption during execution of resource-intensive applications.

To address this challenge, a novel approach called the Bi-generic A2C Microservice Proxy Policy (BAMPP) is proposed. It leverages reinforcement learning (RL) principles to optimize microservice deployment in dynamic Edge-Cloud ecosystems. BAMPP uniquely considers the intricate interdependencies among microservices and adapts to user mobility in real-world scenarios.

© University of Pretoria

This research utilises a simulation platform to reproduce the intelligent edge computing environment, integrating real-world datasets to evaluate the performance of BAMPP against comparative algorithms. The research focuses on three key research points: identifying crucial factors influencing microservice scheduler performance, leveraging RL for optimised scheduling, and assessing the impact of random user mobility on service deployment.

The results demonstrate BAMPP's superior performance in reducing energy consumption, minimizing network usage, decreasing execution and migration latency, and enhancing reliability in microservice scheduling compared to current systems. This research contributes to the field of intelligent edge computing by introducing a novel modeling approach, developing an advanced algorithm for joint optimization of scheduling and resource management, and providing comprehensive performance evaluations using realistic simulations.

The results of this study have important ramifications for raising the effectiveness and performance of microservice applications in intelligent edge environments, potentially leading to cost savings, enhanced sustainability, and widespread implementation across diverse edge computing scenarios.

© University of Pretoria

LIST OF ABBREVIATIONS

A2C	advantage actor-critic
AI	artificial intelligence
API	application programming interface
BAMPP	Bi-generic A2C Microservice Proxy Policy
CBD	central business district
CHM	cardiovascular health monitoring
DAG	directed acyclic graph
DL	deep learning
DRL	deep reinforcement learning
DRN	drone coordination swarm
ECOM	e-commerce
ECG	electrocardiogram
EI	edge intelligence
FC	fog computing
GCN	graph convolutional network
HPC	high performance computing
ILP	integer linear programming
IoT	internet of things
MDP	Markov decision process
MD-DDPG	multiple buffer deep deterministic policy gradient
MEC	multi-access edge computing
ML	machine learning
MLMA	machine learning in microservices architecture
MSA	microservices architecture
PPO	proximal policy optimisation
QoS	quality of service
RL	reinforcement learning
RMS-DDPG	reward memory shaping deep deterministic policy gradient
RSDQL	reward sharing deep Q learning
UE	user equipment

© University of Pretoria

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION **1**
1.1 PROBLEM STATEMENT 1
1.1.1 Context of the problem 1
1.1.2 Research gap 3
1.2 RESEARCH OBJECTIVES AND QUESTIONS 4
1.3 APPROACH 5
1.4 RESEARCH CONTRIBUTION AND NOVELTY 6
1.5 RESEARCH OUTPUTS 7
1.6 OVERVIEW OF STUDY 8

CHAPTER 2 LITERATURE STUDY **9**
2.1 CHAPTER OVERVIEW 9
2.2 MICROSERVICES 10
2.2.1 Architecture deployment 11
2.2.2 Virtualisation 13
2.3 EDGE COMPUTING 17
2.3.1 Platform integrations 20
2.3.2 Performance analysis of edge computing 22
2.3.3 Intelligent edge computing 24
2.4 ARTIFICIAL INTELLIGENCE 27
2.4.1 Overview of AI in edge computing 28
2.4.2 Machine learning 29
2.4.3 Reinforcement learning 31
2.5 CONCLUSION 35

CHAPTER 3 METHODS **38**

© University of Pretoria

3.1 CHAPTER OVERVIEW . . . 38
3.2 DESIGN CONSIDERATIONS . . . 39
3.3 LIMITATIONS . . . 40
3.4 SIMULATOR . . . 41
3.4.1 EdgeAISim . . . 41
3.4.2 Third-party software . . . 43
3.4.3 Dataset . . . 45
3.5 MICROSERVICE APPLICATIONS . . . 47
3.5.1 Cardiovascular health monitoring application . . . 48
3.5.2 Drone coordination swarm application . . . 49
3.5.3 E-commerce application . . . 52
3.5.4 General operational framework . . . 57
3.6 PERFORMANCE MEASURES . . . 59
3.6.1 Quality of service . . . 59
3.6.2 Network usage . . . 61
3.6.3 Energy consumption . . . 62
3.7 MODELS . . . 63
3.7.1 Integer linear programming . . . 64
3.7.2 Advantage-actor critic . . . 67
3.7.3 Proximal policy optimisation . . . 67
3.7.4 Bi-generic A2C microservice proxy policy model . . . 69
3.8 CONCLUSION . . . 71

CHAPTER 4 RESULTS . . . 74
4.1 CHAPTER OVERVIEW . . . 74
4.2 SIMULATION SETUP . . . 74
4.3 DEPLOYMENT OF CHM APPLICATION . . . 76
4.4 DEPLOYMENT OF DRN APPLICATION . . . 80
4.5 DEPLOYMENT OF ECOM APPLICATION . . . 85
4.6 CONCLUSION . . . 89

CHAPTER 5 DISCUSSION . . . 90
5.1 CHAPTER OVERVIEW . . . 90
5.2 DEPLOYMENT OF CHM APPLICATION . . . 90

© University of Pretoria

5.2.1 Network usage and execution time . . . 91
5.2.2 Migration delay . . . 91
5.2.3 Energy consumption . . . 92
5.2.4 Reliability . . . 93
5.2.5 Conclusion . . . 94
5.3 DEPLOYMENT OF DRN APPLICATION . . . 94
5.3.1 Network usage and execution time . . . 95
5.3.2 Migration delay . . . 96
5.3.3 Energy consumption . . . 96
5.3.4 Reliability . . . 97
5.3.5 Conclusion . . . 98
5.4 DEPLOYMENT OF ECOM APPLICATION . . . 98
5.4.1 Network usage and execution time . . . 99
5.4.2 Migration delay . . . 100
5.4.3 Energy consumption . . . 100
5.4.4 Reliability . . . 101
5.4.5 Conclusion . . . 102
5.5 THEORETICAL IMPLICATIONS OF BAMPP . . . 102
5.6 CONCLUSION . . . 104

CHAPTER 6 CONCLUSION . . . 105
6.1 SUMMARY OF FINDINGS . . . 105
6.2 CONCLUSIONS . . . 105
6.3 SUMMARY OF CONTRIBUTIONS . . . 106
6.4 FUTURE WORK . . . 106

REFERENCES . . . 109

APPENDIX A MICROSERVICES SIMULATION PARAMETERS . . . 117

APPENDIX B NUMERICAL RESULTS . . . 120

© University of Pretoria

CHAPTER 1 INTRODUCTION

1.1 PROBLEM STATEMENT

1.1.1 Context of the problem

The IoT has led to the development of resource-intensive applications such as live streaming services, mobile gaming, and virtual reality experiences [1]. These applications are considered resource-intensive because they require significant computational power, high bandwidth, and often real-time processing capabilities. For instance, live streaming services demand continuous data transmission and processing of high-quality video and audio in real-time. Mobile gaming, especially with advanced graphics and multiplayer features, requires rapid computations and constant network connectivity. Also, virtual reality experiences necessitate complex 3D rendering, motion tracking, and minimal latency to maintain user immersion and prevent discomfort. The IoT has transformed how we engage with the world around us. By connecting billions of devices to the internet, IoT has opened up a plethora of possibilities for innovation and technological advancement. Cloud computing was initially recommended to address the lack of computational resources to support the IoT, but concerns over communication cost and latency have prompted the proposal of a complementary infrastructure known as edge computing [2]. Edge computing addresses significant challenges of the existing IoT architecture, such as resource management, service latency, and data privacy. Specifically, edge computing devices handle some computational tasks locally, reducing the burden on centralised cloud computing resources and optimising overall system performance. In terms of service latency, by processing data closer to its source, which could be the home, office, or city, edge computing significantly reduces the time required for data to travel to a centralised remote data centre and back. Edge computing devices processing data locally reduces the amount of potentially vulnerable information that gets transmitted over networks. This helps in complying with data privacy and protection regulations. However, edge computing operates under significant resource constraints compared to cloud computing infrastructure.

© University of Pretoria

A good example is that most edge computing devices are designed to be compact and unobtrusive, limiting the amount of hardware that can be included. Cloud computing infrastructure, housed in large remote data centres, doesn't face such physical constraints. Research on edge computing frameworks is increasing due to advancements in network end and network edge nodes or devices and technological architectures.

Cloud computing, despite its extensive computational resources, faces challenges in providing low-latency response times for IoT applications. To address these limitations, edge computing emerged as a complementary approach. Recognising the constraints of edge computing devices, researchers have integrated two key technologies to enhance edge computing capabilities: artificial intelligence (AI) and microservices. These technologies work synergistically to overcome the resource and performance limitations of traditional edge computing infrastructure. In this sense, AI refers to algorithms and conceptual frameworks that let platforms learn from data and make wise judgements, improving edge device capabilities. Microservices, in contrast, present an architectural approach that decomposes applications into smaller, autonomous services. These services can be created, deployed, and scaled independently, enhancing flexibility and resource utilization. AI and microservices, when integrated with edge computing, provide a solution to unique challenges of IoT such as interoperability among applications, hardware, and resource management to ensure quality user experience and engagement. These inclusive technologies, together with edge computing, are referred to as intelligent edge computing [1]. However, this architectural shift also means the edge computing framework inherits the limitations and constraints of these technologies. AI is a resource-hungry technology known to cause over-utilisation of resources such as energy. At the same time, microservices can result in prolonged or delayed responses, particularly in applications with complex or large dependencies among the microservices [3].

While the integration of AI and microservices with edge computing promises to address several IoT challenges, this architectural paradigm also introduces new complexities. The combination of these technologies, while powerful, creates unique operational constraints that must be carefully managed. The key challenges relevant to this research include:

1. Resource management – Edge computing devices have limited computational resources compared to cloud infrastructure, necessitating the need for efficient allocation strategies.

© University of Pretoria

2. QoS – While edge computing aims to reduce latency and increase reliability, complex dependencies between microservices can still lead to delayed responses. These elements have a direct bearing on QoS, which is the term used to describe how well a network functions as perceived by its users. In the context of this research, QoS typically encompasses metrics such as response time, throughput, availability, and reliability of services.
3. Microservice dependencies – The distributed nature of microservices introduces operational complexities that can impact overall system performance.

To meet the diverse demands of users and address the challenges of the IoT, it is crucial to ensure the harmonious coexistence of various technologies.

1.1.2 Research gap

The scheduling of microservice applications for intelligent edge computing presents a significant challenge. Scheduling refers to the process of deciding where and when to deploy microservices within an intelligent edge computing environment. This involves assigning microservices to specific edge devices or servers, taking into account factors like resource availability, network conditions, and the inter-dependencies between microservices. The goal of scheduling is to optimise the overall performance of the system, ensuring that microservices are deployed in a way that minimises latency, maximises resource utilisation, and meets the QoS requirements of the applications. In event-driven mobility scenarios, deploying microservice applications with a scheduler necessitates the optimisation of competing performance elements, including latency, reliability, and resource consumption, within the intelligent edge computing environment. Meanwhile, microservice applications, due to their operational complexity caused by their distributed architecture and inter-dependencies, have many potential points of failure. This can impact the QoS in terms of latency and communication costs during the deployment of the application.

To address this joint optimisation problem of resource allocation and microservice scheduling in the intelligent edge computing environment, several solutions have been proposed. Multi-objective optimisation approaches using nature-inspired algorithms [4], linear programming [5], and machine learning (ML) [6] have shown promising results in minimising network latency, enhancing service reliability, and optimising resource usage. Other studies considered enabling the hardware of the edge computing devices with secondary features to enhance their performance [7].

© University of Pretoria

While existing research on microservice scheduling in intelligent edge computing environments has demonstrated several critical limitations. Most studies have focused narrowly on either resource allocation or microservice scheduling, rarely addressing both simultaneously. Current approaches often neglect the complex inter-dependencies between microservices, creating significant operational challenges. Moreover, existing methods struggle to adapt to the dynamic nature of mobility scenarios, particularly in environments with rapidly changing network conditions and user mobility patterns.

These limitations create substantial obstacles to achieving optimal microservice deployment within intelligent edge computing environments. Researchers have attempted various optimisation techniques, but these approaches fail to comprehensively address multiple performance objectives concurrently. The need for a holistic approach that can dynamically manage microservice scheduling, resource allocation, and performance optimisation remains a critical gap in the current research landscape.

1.2 RESEARCH OBJECTIVES AND QUESTIONS

This dissertation proposes that an RL-based technique can significantly improve the efficiency of microservice scheduling and resource allocation in intelligent edge computing, thereby enhancing QoS metrics including latency, network bandwidth utilisation, and energy consumption, while accounting for the complex inter-dependencies of microservices in dynamic, real-world scenarios. This novel technique, called BAMPP, will meticulously account for the intricate inter-dependencies among microservices, the fluid nature of the environment, and the paramount need to optimise multiple performance objectives. Through BAMPP, this research aims to establish a robust and adaptive microservice scheduling strategy that can efficiently handle the complexities of intelligent edge computing. The research questions addressed include:

1. What are the key factors that influence the performance of a microservice scheduler in the intelligent edge computing environment?
2. What methods have been implemented to address the performance decline of microservice schedulers in the intelligent edge computing environment?
3. How can RL be leveraged to ensure microservice scheduling that optimises latency, network bandwidth, and energy usage?

© University of Pretoria

1.3 APPROACH

The approach of this research is to reproduce the intelligent edge computing environment on a simulation platform. This involved creating a virtual representation of the edge computing environment, including the hardware, software, and network components. The simulation platform allowed for the experimentation and evaluation of different edge computing algorithms and architectures in a controlled environment.

Additionally, the research integrated a real-world data set into the simulation environment. This data set consists of real-world data that is relevant to the research objective. By integrating this data set, the research evaluates the performance of the BAMPP algorithm and comparative algorithms on real-world data.

The approach considered the following key factors:

1. Inter-dependencies of microservices – Microservices are not isolated entities; they often depend on each other to function correctly. These inter-dependencies can create challenges when deploying microservices, as the deployment order and configuration of one microservice can impact the performance of others. The proposed RL-based approach considers these inter-dependencies when making scheduling decisions, ensuring that the overall system is optimised.
2. Dynamic nature of the environment – The intelligent edge computing environment is dynamic, with factors such as network conditions, resource availability, and user demand constantly changing. This dynamism can make it difficult to make static scheduling decisions. The proposed RL-based approach adapts to these changes in real-time, ensuring that microservices are deployed in the most optimal way possible.
3. Optimization of multiple performance objectives – When scheduling microservices, it is important to consider multiple performance objectives, such as latency and bandwidth usage. These objectives can often be conflicting, making it difficult to find a deployment configuration that optimizes all of them. The proposed RL-based approach learns from experience and finds the best trade-off between these objectives, ensuring that the deployed system meets the desired performance requirements.

To address these challenges, the BAMPP algorithm is characterised by:

© University of Pretoria

1. Joint-optimization – BAMPP simultaneously optimises microservice scheduling and resource allocation, addressing the complex interaction between these factors.
2. Adaptive learning – BAMPP employs the actor-critic policy gradient optimisation to learn and adapt to the dynamic nature of intelligent edge computing environments in real-time.
3. Microservice dependency awareness – BAMPP explicitly considers the inter-dependencies between microservices, ensuring optimal placement that minimises communication overhead and latency.

The proposed RL-based approach is evaluated using a variety of simulation settings and a real-world dataset. The trials' outcomes serve as evidence for the suggested approach's effectiveness in maximising microservice adoption in the intelligent edge environment.

1.4 RESEARCH CONTRIBUTION AND NOVELTY

This research makes the following technical contributions to the field of intelligent edge computing:

1. The novel approach models the complex intelligent edge computing environment as an RL environment. This enables more accurate simulation and analysis of microservice dependencies. By leveraging the principles of RL, this research aims to capture the dynamic and interactive nature of the intelligent edge computing environment. This allows a better understanding of the impact of various factors, such as network conditions, resource availability, and workload patterns, on microservice performance.
2. The creation of the BAMPP algorithm is a noteworthy breakthrough for solving the joint optimisation problem of microservice scheduling and resource management in intelligent edge computing environments. Unlike existing solutions that often focus on one aspect or the other, BAMPP considers both scheduling and resource allocation decisions simultaneously. This holistic approach leads to improved overall performance, as it ensures that microservices are not only placed on the most suitable edge computing nodes but also allocated the necessary resources to meet their performance requirements.
3. To evaluate the effectiveness of BAMPP, simulations are conducted using EdgeAISim, an AI-based environment for simulating edge computing scenarios. EdgeAISim enables the simulation of real-world datasets and applications, providing a more realistic evaluation of intelligent edge computing behaviour during microservice application deployment. The use of EdgeAISim

© University of Pretoria

allowed the assessment of BAMPP's performance across a range of scenarios, including different network conditions, resource constraints, and workload patterns.

4. The simulations showed that BAMPP performed better than the most advanced techniques, particularly in achieving optimal microservice performance in intelligent edge environments. BAMPP consistently outperformed other algorithms in terms of key metrics such as latency, throughput, and resource utilization. These results have important implications for the field, as they suggest that BAMPP can significantly improve the performance and efficiency of microservice applications. Potential benefits include cost savings through reduced infrastructure requirements, enhanced sustainability through improved energy efficiency, and the possibility of wide-ranging implementation across diverse edge computing scenarios.

In general, this research is significant because it addresses critical challenges in the rapidly evolving field of edge computing and IoT. By improving the efficiency of microservice deployment and resource allocation, this research has the potential to enhance the performance of IoT applications across various domains, from smart cities to industrial automation. Moreover, the proposed approach could lead to substantial energy savings and cost reductions in large-scale IoT deployments, contributing to both economic and environmental sustainability.

1.5 RESEARCH OUTPUTS

Currently, a conference paper entitled "Comparative Analysis of Existing Algorithms for Energy-Efficient Microservice Placement in Edge Computing Networks" is in progress. This research has contributed to the following publication in a peer-reviewed journal.

1. K. Afachao, A. M. Abu-Mahfouz, and G. P. Hanke, "Efficient microservice deployment in the edge-cloud networks with policy-gradient reinforcement learning," *IEEE Access*, vol. 12, pp. 133 110–133 124, Sep. 2024.

Several conference papers have been published in reference to this research. These published papers include:

1. K. Afachao and A. M. Abu-Mahfouz, "Towards energy-efficient intelligent edge computing," in *Proc. Int. Conf. Elect. Comput. Energy Technol.*, Nov. 2023, pp. 1–6.

© University of Pretoria

2. K. Afachao and A. M. Abu-Mahfouz, "Running virtual services for the intelligent edge: A review," in *Proc. IEEE World AI IoT Congr.*, Jun. 2023, pp. 0204–0209.

3. K. Afachao and A. M. Abu-Mahfouz, "An examination of virtualization technologies for enabling intelligent edge computing," in *Proc. Int. Conf. Sustain. Comput. Smart Syst.*, Jun. 2023, pp. 1612–1617.

1.6 OVERVIEW OF STUDY

This dissertation is structured into the following chapters. Chapter 2 provides a comprehensive review of existing literature on microservices, edge computing, and AI. It explores the evolution of these technologies, their current state, and the challenges associated with their integration. In Chapter 3 the details of the research methods employed in the study are described, including the design considerations, simulation setup, dataset used, and performance metrics. It describes the implementation of the proposed BAMPP model and the state-of-the-art algorithms used for comparison. Chapter 4 presents the results of the simulation experiments, comparing the performance of BAMPP with the other algorithms across various performance metrics. It discusses the findings and their implications for microservice deployment in intelligent edge environments. In Chapter 5 the results are analysed in detail, interpreting the findings and relating them to the research questions. It discusses the strengths and limitations of the proposed approach. Chapter 6 provides an overview of the research's major findings and makes inferences from them. It highlights the contributions of the research to the field of intelligent edge computing and outlines potential directions for future work.

© University of Pretoria

CHAPTER 2 LITERATURE STUDY

2.1 CHAPTER OVERVIEW

This literature review provides a comprehensive examination of the technological landscape surrounding intelligent edge computing, with a focus on its integration with microservices and AI. The chapter is structured to progress from foundational concepts to aspects reflecting the nature of this research.

Section 2.2 introduces microservices, exploring their architecture, benefits and challenges. This section also delves into virtualisation and technologies and deployment strategies, which are crucial for understanding the implementation of microservices in edge computing environments.

Section 2.3 focuses on edge computing, tracing its evolution from traditional cloud computing architectures. It examines various edge computing paradigms, analyses performance considerations, and explores platform integrations. This section aims to highlight the unique challenges and opportunities presented by edge computing environments.

Section 2.4 delves into the application of AI techniques in edge computing, with particular emphasis on RL. This section discusses how AI can optimise resource management, improve decision-making processes, and enhance overall system performance in edge computing environments.

The literature was selected through a systematic search of peer-reviewed journals, conference proceedings, and industry reports published between 2013 and 2024. The focus was on works that specifically addressed the intersection of microservices, edge computing, and AI, in any form or manner. Priority was given to comprehensive and well-cited papers.

© University of Pretoria

By the end of this chapter, readers should have a clear understanding of the current landscape, challenges and potential future developments of intelligent edge computing as it relates to optimising microservice scheduling and resource management in edge computing environments through AI techniques.

2.2 MICROSERVICES

Microservice architecture (MSA) is a software development architectural style that addresses the limitations of monolithic applications such as poor scalability. It is composed of microservices as the basic units, which are unified, independent processes that interact via application programming interface (API) calls. One good example of a microservice application is the Netflix application [12]. In contrast, monoliths are applications whose composure cannot be executed independently, a good example of such an application is Istio [13].

Software architecture, the design pattern driving a software system's reliable performance, evolved through the challenges of large-scale development in the twentieth century, culminating in microservices' emergence. The 1970s sparked interest in software architecture and design, leading to the integration of development processes in the 1980s and 1990s. Object orientation influenced software architecture, and the advent of service-oriented computing paved the way for the microservices concept introduced in 2011. Although relatively new, microservices gained attraction in prominent companies like Netflix and Amazon, leveraging their benefits to enhance user experiences and drive innovation.

The challenges of maintaining and evolving large monolithic systems are multifaceted and present significant obstacles to their effective management. One of the primary challenges lies in the difficulty of scaling the application due to dependency conflicts. Dependency conflicts arise when different components of the monolithic system rely on different versions of the same library or framework, leading to compatibility issues and potential errors. As the system grows and evolves, managing these dependencies becomes increasingly complex, requiring careful coordination and testing to ensure that all components work harmoniously.

Another challenge associated with monolithic systems is the limitation that the programming language remains the same throughout the application's lifetime. This can hinder the ability to adopt new and improved programming languages, frameworks, and libraries that may offer better performance,

© University of Pretoria

security, or ease of development. Additionally, it can make it difficult to attract developers who are proficient in other programming languages.

The monolithic architecture also presents challenges in terms of modularity and code reusability. Changes made to one part of the system can have unintended consequences in other parts, making it difficult to isolate and fix issues. This can lead to a cascading effect of errors and a high risk of introducing new bugs. Furthermore, the lack of modularity makes it challenging to reuse code across different parts of the application, resulting in duplication of effort and increased development time and cost.

Finally, monolithic systems can be challenging to deploy and update, as changes to the system require redeploying the entire application. This can lead to downtime and disruption of services, particularly for large and complex systems. Additionally, testing and verifying changes in a monolithic system can be time-consuming and error-prone, as it requires testing all the interconnected components to ensure that they work correctly together. In contrast, microservices offer benefits such as ease of maintenance, improved scalability, and technology flexibility, as various programming languages can be employed and communicate with each other via API messages.

While initial research emphasises the advantages of MSA, a more comprehensive analysis is necessary to evaluate the trade-offs associated with its adoption. This analysis would include an assessment of increased operational complexity and potential performance overhead. Furthermore, there is a need for quantitative comparisons between microservices and alternative architectural styles, which are essential for comprehending the relative advantages and disadvantages of MSA in detail. These comparisons would help to clarify the relative advantages and disadvantages of microservices, enabling architects to make informed decisions about the suitability of MSA for their specific requirements.

2.2.1 Architecture deployment

Advancements in computing capabilities have been significantly driven by technology companies. Some of these advancements have been made possible through the provision of remote data centres, known as cloud data centres, which are equipped with vast amounts of resources. With these data centres, teams, developers, and enterprises access computing power, making it easier to realise projects and application designs. Cloud computing services are typically offered as infrastructure as a service, platform as a service, or software as a service, giving users flexible options to meet their specific

© University of Pretoria

needs. The cloud has benefited from years of research and development which have equipped cloud computing with features now commonly referred to as cloud-native traits. These cloud-native traits are microservices, container-based virtualisation and other virtualisation technologies such as network function virtualization [14]. Despite these strengths the cloud computing infrastructure is inadequate to deal especially with the demands for the IoT use cases.

Sun et al. [15] predicted the decline of centralised cloud computing services as the exclusive resource provider for microservices and a means to meet the IoT need for scalability and flexibility. They proposed a general system design that enables interoperability and accommodates diverse devices, without placing their work in the context of computing architectures. Their system design prioritises core service provision and facilitates rapid device communication. They intended to create a system architecture that would remain valuable regardless of future technological advancements.

Pallewatta et al. [16] proposed fog computing (FC), a reference architecture of edge computing as a paradigm to expand cloud computing services to the network's edge. They argued that FC is an ideal environment for running microservice applications due to its distributed, heterogeneous, and resource-constrained devices. Their environment minimised latency and network usage by leveraging decentralised nodes during service placement.

The microservices architecture aligns with the decentralised nature of FC environments, as it is independently deployable and scalable. Service placement refers to optimising scheduling tasks and mapping them to devices while ensuring resource limitations are fulfilled. This is a three-part problem, unlike Fazio et al.'s two-part problem of scheduling tasks while guaranteeing the fulfilment of resource limitations. Through a simulation for a singular microservices application, the authors demonstrated the superiority of fog-based microservices deployment over cloud-based microservices in terms of latency and network usage. Their results revealed an 85% improvement in both latency and network usage for fog-based microservices compared to the cloud-based alternative.

However, further research is necessary to determine the effectiveness of their proposed policy in extremely resource-constrained environments and across multiple metrics beyond latency and network usage.

© University of Pretoria

2.2.2 Virtualisation

The widespread adoption of microservices can be largely attributed to the emergence of virtualisation technologies. In computing, these technologies provide an abstraction layer for application environments. This concept is closely linked to high performance computing (HPC) environments, particularly cloud computing, where virtualisation extends hardware resources such as storage, memory, and compute to meet the demands of a growing user base. The increasing importance of virtualisation stems from concerns about energy consumption and carbon emissions from data centres. Virtualisation enhances system flexibility and reduces power consumption by managing resources efficiently and turning off idle machines while maintaining performance requirements [17]. Microservices are instantiated at the operating system level through container-based virtualisation engines. Container-based virtualisation involves partitioning the physical machine's resources to create isolated user-space instances known as containers. Each container functions like a standalone operating system, sharing the host's kernel but maintaining its file system, processes, and network interfaces. Key advantages of this virtualisation form include close-to-native performance, efficient resource utilisation, and rapid provisioning. It is particularly beneficial in HPC environments like cloud computing, where it incurs less overhead cost compared to traditional virtualisation known as hypervisor-based virtualisation. Container-based virtualisation offers microservices better resource sharing and custom environments without the significant performance overhead associated with traditional virtualisation called hypervisors, making it an attractive option for HPC applications [18]. Figure 2.1 shows how these architectures relate to each other.

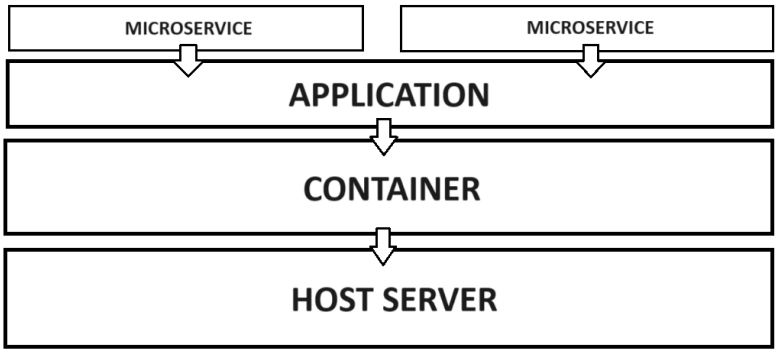

Figure 2.1. A visual representation of how microservices relate to containers and a computing server.

© University of Pretoria

Fazio et al. [19] analyse the relationship between microservices and virtualisation technologies in their research. They focus on how microservices can be deployed using container-based virtualisation. Containers, they explain, are collections of operating system utilities that allocate and manage computing resources for application components. This approach allows cloud computing providers to provision and optimise resources with flexibility. The authors contrast this method with traditional hypervisor-based virtualisation, emphasising that containers have lower overheads due to their shared operating system kernel. However, this also results in reduced isolation, which can increase security risks. Additionally, they note that containers are well-suited for cloud computing environments, although this observation was made before the recognition of edge computing's capabilities for HPC.

The authors identify several challenges in scheduling microservices while managing computational resources. First, they highlight the difficulty in implementing a holistic monitoring mechanism that provides real-time system performance based on recognised metrics. Second, they note the complexity of handling diverse configurations of microservices across the computing environment. These challenges stem from the nature of microservices as unified, independent units with divergent runtime needs. Some microservices are compute-heavy while others cause high communication overheads due to their reliance on communication.

In their study, Hassan et al. [20] address the research gap in the transition from monolithic applications to microservice applications by providing a thorough analysis of the current state and practice of microservices in the industry. They highlight the lack of quantitative results to validate this transition. The authors acknowledge that the contemporary term "microservices" is often used interchangeably with service-oriented architectures in informal literature but note a lack of consistency between industry and research regarding the standard terminology. However, the paper's reliance on industry reports and informal literature limits its academic rigour.

Despite this limitation, the authors propose a self-adaptive solution to mitigate the impact of microservice granularity on the performance of a hypothetical movie streaming application. This solution has the potential to lead to more cost-effective and maintainable microservice architecture. While the paper discusses the potential benefits of self-adaptivity in microservices, it does not explore the practical implications of implementing such a solution through experimentation. Gan et al. [21] address a crucial research gap: the absence of a comprehensive analysis of microservice applications in practice and their implications. They developed an open-source benchmark suite for microservices, considering

© University of Pretoria

their performance on real-world end-to-end applications such as social networks, media services, banking systems, and IoT applications. Additionally, they assess the performance of the applications over metrics such as resource utilisation and QoS.

While the authors do not explicitly explain why the IoT application was implemented on edge computing devices rather than solely on cloud computing devices, they demonstrate that without edge computing supporting cloud computing for low computational applications, latency increases exponentially. However, with the addition of edge computing devices, there is a significant improvement in latency.

In terms of service delivery, one way to measure user satisfaction is through QoS measures. QoS encompasses various factors, and since user perception of service can vary, latency measures provide a close approximation. Latency refers to the time it takes for an application to respond under normal conditions. The authors consider QoS in terms of latency and resource utilisation. They emphasise that the varying performance of microservices during runtime is difficult to predict and maintain using deterministic methods. Furthermore, the authors report difficulty in monitoring the performance of the microservices from the client–side only, especially in determining which microservices are responsible for a decline in QoS. The cause of this problem is that microservices' dependencies can create cascading QoS violations, leading to system-wide end-to-end failures which provide vague interpretability from the client side. The cost of this unpredictability can lead to severe downtimes. This challenge was anticipated by Hassan et al. but attempts to realise their vision were not feasible at the time due to the preference for simple heuristic approaches to ensure microservices' adaptability to resource constraints. Another problem worthy of mention is the general performance decline experienced as the complex dependencies of microservices increase. The more dependencies there are, the more relationships amongst them cause contention.

In their study, Tapia et al. [22] aim to provide quantitative performance results for microservices, comparing them to other architectures. Unlike Gan et al., who considered a range of applications, they focus on a web application with chat, commenting, and notification features, using both monolithic and microservice architectures. The authors acknowledge that this application may not be representative of all applications, as Gan et al. found that microservices' performance varies depending on the application and its dependencies. However, they intended to explore the strengths and weaknesses of each architecture for different applications in the future. Their preliminary study suggests that

© University of Pretoria

both monolithic and microservice architectures have limitations, and companies should tailor their choice to their specific business needs. While monolithic architecture is known for its security and broad applicability, it may not be as effective as microservices in handling rapid growth and continuous integration. The comparison reveals that microservices tend to consume more CPU and network resources than monolithic architecture, while the latter requires more memory and disk writing resources. These trade-offs can affect system performance, depending on the available resources of the device. Based on these findings, the authors favour microservices as the better choice.

Microservices may not be the best choice for an application in certain scenarios. Gravanis et al. [23] highlight that the trend of adopting microservices can lead to unnecessary complexity and cost, particularly when a traditional monolith architecture may still be suitable. This assertion was supported by Blinowski et al. [24], who found that a monolith can outperform a microservice-based architecture on a single machine. When choosing between MSA and monolithic architecture, they advised that organisations should carefully consider certain factors. Complex systems are more suited for MSA, while simple systems may not require the added complexity of microservices. Legacy systems are often better candidates for monolithic architecture, as refactoring them into microservices can be challenging. Applications that are likely to grow and expand in the future are better suited for MSA, as it allows for easier scalability and flexibility. Organisations with limited resources and small teams should consider monolithic architecture, as it requires less overheads and maintenance than MSA. To further emphasise the point, Mendonça et al. [13] presents a case study of Istio, a company that migrated from microservices to a monolithic architecture. The Istio operators became disenchanted with microservices after failing to observe the promised advantages. Their initial adoption of microservices was based solely on industry best practices. However, after re-evaluating their application, they realised that a monolithic architecture was a more suitable choice due to their application's simplicity.

While researchers have focused on microservices and monolithic architectures, serverless functions, under serverless computing has emerged as a new approach in software development. It offers benefits like easier scalability by transferring server management to cloud providers. There is not a clear consensus on whether serverless functions are superior to microservices. Tusa et al. [25] compared microservices and serverless functions in edge computing, focusing on scalability, data processing, speed, and resource usage. They found that serverless functions scaled better, handling up to 250,000 messages per second. However, microservices were more resource-efficient under consistent workloads. Allen et al. [26] conducted a similar study in cloud computing and confirmed

© University of Pretoria

that serverless applications performed better with increasing traffic, while microservices' performance declined. However, serverless functions also had higher overhead, impacting performance with multiple parallel requests. Further research is needed to understand how interactions between microservices and serverless functions impact performance. This granular analysis will provide deeper insights into the strengths and weaknesses of each approach.

Despite progress in addressing research gaps related to microservices and virtualisation, evaluating the operational complexity of microservices and the overhead cost incurred during runtime remains a challenge. Future research, including this research, aims to investigate this area comprehensively.

While this section review provides a comprehensive overview of microservices architecture, its deployment, and its relationship with virtualisation technologies, it's important to note that future research could expand on these areas to provide a more holistic view of microservices architecture and its practical implications.

In the next section, consideration is given to the environment in which these services often operate: edge computing. The principles of MSA, particularly their modularity and scalability, become especially relevant in the distributed and often resource-constrained environment of edge computing. The challenges of deploying microservices are amplified in edge computing environments, necessitating novel approaches to resource management and performance optimisation.

2.3 EDGE COMPUTING

In the digital landscape, the IoT serves as a cornerstone, enabling seamless interconnectivity of physical objects with the Internet. This advanced technology empowers devices to identify themselves to other devices and exchange data effortlessly. IoT seamlessly merges the physical world with computerised systems, resulting in increased coordination, refined data control, and lucrative economic incentives driving the growth of the technology industry.

However, the emergence of IoT challenged cloud computing's dominance as the sole resource provider. This necessitated a shift from cloud computing to a new architecture called edge computing. In order to reduce latency and maximise bandwidth, edge computing deliberately places processing and data storage less distant from the point of demand.

© University of Pretoria

In the early days of the internet, cluster computing, a precursor to edge computing, existed [27]. However, with the advent of cloud computing in the early 2000s and the need to process data closer to data sources for IoT, modern edge computing emerged. It became not only an extension of cloud computing but also inherited some characteristics of cluster computing, such as consolidating heterogeneous devices for increased resource strength.

The transition from cluster computing to cloud computing and then to edge computing was not solely driven by IoT. Various factors, such as the need to address challenges like high latency, jitter, lack of location-awareness during user mobility, and non-interoperable communication types in cloud computing paradigms when serving real-time applications, played a significant role in this evolution.

Several edge computing paradigms have emerged to shift digital services from centralised cloud computing to the network edges, closer to where the data is generated. These include FC, multi-access edge computing (MEC), and Cloudlet. While similar in their overall goal, these paradigms differ in their specific application requirements and implementation approaches. Table 2.1 shows a simple overview of how they compare.

MEC is a key technology in 5G systems, providing mobile random access network application solutions. This method works especially well for applications like augmented reality and driverless cars that need high bandwidth and low latency [28].

In contrast, IoT devices can analyse data at the edge of the network thanks to FC's direct resource extension. FC is the perfect platform for IoT applications that need mobility and location awareness since it can handle a wide range of devices, including base stations and single-board computers. FC includes various connected resource-constrained devices like smartphones, smartwatches, and routers [27].

Cloudlets are devices positioned between edge devices and cloud computing. They offer a device one hop away from users for computation offloading from mobile devices. Devices with moderate capabilities, such as network cell sites and local area networks, are part of Cloudlet and MEC infrastructure.

© University of Pretoria

Table 2.1. This table provides a comparison of the three main edge computing reference architectures.

Feature	**FC**	**MEC**	**Cloudlet**
Standardisation	OpenFog Consortium	ETSI MEC	Open Edge Computing Initiative
Primary Focus	IoT devices	Mobile networks	Mobile device
Typical Use Cases	Smart homes, Smart cities	Virtual reality, Autonomous vehicles	Content delivery

Sitton-Candenedo et al. [29] reviewed edge computing infrastructure, highlighting the conflicting nature of reference architectures. They analysed emerging architectures such as Intel-SAP, FAR-Edge Project, and the Industrial Internet Consortium's Edge Computing RA for Industry 4.0. Through literature surveys, they identified limitations in existing architectures, including the lack of standardised traits in the edge computing environment. These architectures find it difficult to handle the complexity of solutions for smart homes and cities in addition to Industry 4.0 due to the presence of diverse standards and proprietary hardware and software.

In response, the authors proposed a global reference architecture for edge computing and demonstrated its feasibility through a real-world scenario in the smart agriculture industry. Their architecture incorporates an IoT platform enhanced with blockchain for security and privacy, as well as modularity to handle interoperable dependencies.

However, their research has several shortcomings. Their proposed architecture, implemented through a smart agriculture industry scenario, lacks adequate performance evaluations and a discussion on the implications and effectiveness of the results. In summary, the authors provide limited evidence for the feasibility of this architecture, primarily focusing on its theoretical viability without addressing practical considerations.

Although their research had limitations, they established a future research path for edge computing that aims to achieve a holistic paradigm that is robust, secure, and sustainable.

© University of Pretoria

2.3.1 Platform integrations

Over time, cloud computing has undergone significant research and development, acquiring cloud-native traits such as microservices and container-based virtualisation. In contrast, edge computing still lacks autonomous sustaining systems. This disparity highlights one of the most significant challenges facing the edge computing paradigm.

Malazi et al. [30] envision a future for edge computing that seamlessly integrates cloud-native technologies like microservices, blockchain, ML, and data analytics within edge-cloud environments. These technologies, they believe, hold great promise for optimising performance, bolstering security, and safeguarding privacy for edge computing environments.

Microservices, through virtualisation, enable efficient utilisation of hardware resources by allowing multiple virtual machines to operate on a single physical device. This flexibility optimises resource allocation within edge computing environments.

ML models leverage sensor data analysis to predict equipment failures proactively. This enables proactive maintenance strategies and minimises downtime, resulting in increased operational efficiency and cost savings.

Edge computing data analytics processes raw data at the network edge, reducing its volume and minimising the need for constant transmission to remote data centres. This conserves bandwidth, optimises network performance, and reduces latency, making it particularly valuable in resource-constrained environments.

Vano et al. [14] acknowledge the benefits of edge computing and the increasing interest in applying cloud-native design principles to enhance its adaptability, affordability, and user accessibility. However, they identified several challenges associated with implementing these technologies at the network edge. These challenges stem from the resource-constrained nature of edge computing environments and the often unreliable network connection. The user accessibility is achieved through close proximity between users and edge devices. This proximity minimises latency, improves bandwidth availability, and enhances the overall responsiveness of applications. By leveraging cloud-native design principles,

© University of Pretoria

they believe edge computing can become more adaptable, affordable, and accessible to a broader range of users.

However, implementing these technologies at the network edge presents several challenges. One significant hurdle lies in the resource-constrained nature of edge computing environments. Edge devices typically have limited processing power, memory, and storage capacity compared to traditional cloud servers. This resource scarcity hinders the execution of complex computations and storage-intensive applications at the network edge.

Another challenge stems from the often unreliable network connection at the network edge. Remote or rural locations with poor or nonexistent internet connectivity are common places to install edge devices. This network unreliability can lead to data loss, service disruptions, and degraded user experiences.

To address these challenges, the authors propose several solutions. They suggest lightweight container orchestration engines, implementing an edge-cloud continuum, and utilising operating systems like EVE-OS and programming languages like Rust at the network edge. These solutions, they believe, would provide a universal, vendor-agnostic architecture.

Building upon the work of Malazi et al. and Vano et al., Le Duc et al. [6] delve deeper into edge computing by exploring technologies and methods that address resource provisioning challenges in edge-cloud environments. They investigate the edge-cloud environment, which combines edge and cloud computing, to leverage the strengths of both approaches. Cloud computing offers optimised performance, while edge computing reduces latency. This aligns with the concept of an edge-cloud continuum introduced by Vano et al.

Le Duc et al. consider technologies such as ML, virtualisation, and optimisation techniques. ML, which was also explored by Malazi et al., is highlighted for its ability to model, predict, and manage complex distributed applications, leading to improved reliability. It excels at anticipating resource demands, enabling proactive scaling and optimisation. Virtualisation efficiently allocates resources across the network, enhancing performance. Optimisation ensures applications can adapt to workload changes and recover from failures, maintaining service quality and user experience. Rather than

© University of Pretoria

presenting their results in charts and figures, the authors review and discuss the findings from other studies in the field.

Samanta et al. [31] align with Le Duc et al. in proposing an advanced automated system. This system incorporates cloud computing, edge computing, microservices, and ML to enhance edge computing infrastructure. The integration of these technologies, they argue, leads to systems that are corrective, responsive, and intelligent. As a result, the QoS for end users is optimised, latency is minimised, and microservice completion time is reduced.

In terms of optimisation, the authors emphasise the importance of a dynamic microservice scheduler. This scheduler adapts to varying network conditions and infrastructure, which is crucial for maximising network throughput and resource utilisation while minimising total network latency.

To evaluate the performance of their proposed model, the authors present extensive results across several metrics such as throughput, QoS, failure rate, resource utilisation, and average price. The results indicate that their model, "dynamic microservice scheduling in edge computing" algorithm, achieves superior network throughput and resource utilisation within optimal delay constraints. This leads to significant improvements in QoS and average price under different network conditions. It is, however, important to note that none of these studies factors the performance cost of introducing microservices dependencies on deployment.

2.3.2 Performance analysis of edge computing

Cloud computing has limitations in meeting the stringent latency, availability, and reliability demands of IoT microservices applications. Alam et al. [32] emphasise the crucial role of edge computing in addressing these challenges. Their findings suggest that a layered architecture utilising both cloud and edge computing can effectively manage resources and enhance system performance.

However, Ali-Eldin et al. [33] delve into the edge computing performance inversion problem, where resource-constrained edge computing servers can sometimes result in higher end-to-end latency than cloud data centres. Edge computing device performance inversion occurs at moderate utilisation levels between 40% and 60%, making cloud computing a better option.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

To address this issue, Scoca et al. [34] concentrate on scheduling latency-aware applications in edge computing environments. Their proposed scheduling algorithm takes into account both network and computational capabilities of edge computing nodes to optimise service placement. Edge computing reduces network delay compared to cloud computing solutions, but limited computational resources can result in longer processing times.

Aral et al. [35] aimed to meet the latency requirements for data-intensive applications in edge computing environments. Their scheduling technique assesses the computational, and reliability capabilities of edge computing nodes. Their hybrid approach, combining edge computing and cloud computing resources, offers the best of both worlds by maintaining low network delay while leveraging the cloud computing's processing power when edge computing nodes are saturated.

Caiazza et al. [36] and Ahvar et al. [37] have conducted significant research on the energy consumption of edge computing compared to centralised cloud computing. Caiazza et al. focused on a request-response communication scheme and found that edge computing can lower client device energy usage by up to 55% compared to cloud computing. This is primarily due to the reduced communication latency when using edge computing servers. Ahvar et al., on the other hand, proposed a taxonomy and a generic energy model to evaluate and compare the energy consumption of cloud, fog, and generic edge computing. Their study revealed that completely distributed architectures such as edge computing are more energy-efficient compared to fully centralised and partly distributed cloud computing architecture.

These studies highlight key insights into edge computing. There exists a consensus that edge computing can significantly reduce latency and network usage compared to centralised cloud computing, as demonstrated by Alam et al. [32] and Caiazza et al. [36]. However, Ali-Eldin et al. [33] revealed an important caveat with the performance inversion problem, indicating that edge computing's benefits are not universal and depend on utilisation levels.

The many studies show how intricately edge computing performance involves trade-offs. While Scoca et al. [34] and Aral et al. [35] propose scheduling algorithms to optimise service placement and meet latency requirements, their approaches must balance competing factors such as network conditions, computational capabilities, and energy efficiency. The energy consumption analysis by Caiazza et al. [36] and Ahvar et al. [37] further underscores this complexity, showing that edge computing can be

© University of Pretoria

more energy-efficient. Still, this efficiency depends on factors like network architecture and application characteristics.

These findings suggest a need for more adaptive, context-aware optimisation strategies in edge computing. Additionally, there is a gap in understanding how these performance characteristics scale in large, heterogeneous edge computing environments. It is critical to solve these restrictions in order to have a more complete picture of edge and cloud computing performance.

2.3.3 Intelligent edge computing

The transition towards edge computing is primarily driven by the necessity for more efficient and low-latency processing. This approach brings data processing closer to its source, eliminating the need to transmit it to distant remote data centres. Consequently, network congestion is alleviated, and service responsiveness is enhanced, making it ideal for time-sensitive applications such as autonomous driving. Additionally, edge computing enables the implementation of ML algorithms that reduce connectivity requirements among devices, provide higher bandwidth availability, and improve data privacy. Intelligent edge computing has emerged as a response to the unique scenario created by the merger of AI and edge computing in handling the massive amounts of data generated by ubiquitous sensors and smart devices [1]. MSA complements the intelligent edge by allowing AI functionalities to be deployed as individual services at the edge, enhancing scalability and manageability in intelligent edge ecosystems.

Intelligent edge computing, also known as edge intelligence (EI), integrates ML and AI with edge computing. This relatively new concept has recently gained significant attention. EI sets itself apart by executing AI algorithms at the network edge, encompassing end devices that generate data.

As a rising paradigm, EI is still the subject of active research. Current literature reveals two main perspectives on its definition:

1. A broad category with two main research directions: AI for edge and AI on edge. AI for edge involves applying AI to optimise edge computing problems, while AI on edge focuses on running AI models on edge devices [38].

© University of Pretoria

2. A crucial part of the larger EI goal, especially meant to make edge computing-based deep learning (DL) service deployment easier [3].

Due to the lack of a universally agreed-upon definition, the terms intelligent edge and EI are often used interchangeably in the field. This fluidity in terminology reflects the evolving nature of this emerging technological paradigm.

The adoption and implementation of EI encounter various challenges. First, efficient programming and software platforms are crucial for the effective delivery of edge computing services. Moreover, standard performance indicators fall short of capturing the runtime performance of deep neural networks (DNN) model inference on edge computing devices, thus necessitating trade-offs between accuracy, inference speed, and resource usage. Another critical challenge lies in service and resource management, encompassing efficient service discovery protocols and distributed edge computing resource orchestration. Furthermore, the high-dimensional configuration problem, involving real-time determination of performance-critical configuration parameters, poses difficulties due to its combinatorial nature and vast search space. Additionally, adapting ML algorithms for distributed computing and storage systems, as well as managing computing and memory resources, present significant challenges.

Several strategies are being employed to optimise AI models for edge computing environments. These strategies aim to maintain high performance while reducing computational complexity and resource usage. Model optimisation techniques include weight matrix sparsification, pruning, weight quantisation, and low-rank decomposition [39]. Weight quantisation, a key approach, involves reducing the precision of model weights from 32-bit floating points to 8-bit integers. This results in a smaller model size and faster inference speed. Additionally, hardware-specific optimisations leverage specialised hardware like tensor processing units and graphics processing units to enhance performance. For example, Google's edge tensor processing units are designed for energy-efficient inference tasks, while NVIDIA's Jetson platform offers scalable computing power for complex applications [40]. Furthermore, the use of analog memory circuits and in-memory computation shows promise for improving speed and energy efficiency. By performing vector-matrix multiplication in the analog domain, significant performance gains can be achieved compared to traditional digital architectures.

Notwithstanding, the integration of AI models with edge computing platforms and microservices is effectively achieved through several key strategies and frameworks. Containerisation technologies,

© University of Pretoria

such as Docker, facilitate the encapsulation of AI models along with their dependencies, enabling easy deployment and management across diverse edge devices. Adopting a microservices architecture allows AI applications to be developed as a collection of loosely coupled services, enhancing flexibility and resilience through independent scaling and updating of components. Edge computing frameworks like KubeEdge and Open Horizon provide the necessary infrastructure for deploying and managing AI workloads at the edge, supporting orchestration and resource management. Federated learning enables collaborative model training across multiple edge devices through gradient sparsification where only significant gradients are transmitted and asynchronous updates, which allow edge nodes to keep data localised while training separate models, thus enhancing privacy and reducing data transfer needs [41]. Additionally, automatic architecture search methods are employed to optimise network structures for specific tasks and hardware constraints [42]. Together, these strategies and frameworks foster a seamless integration of AI with edge platforms and microservices, striving to promote efficient deployment and scalability in edge computing.

The future of intelligent edge computing holds promising advancements across several key areas: real-time data processing, enhanced privacy and security through federated learning, optimised resource management and adaptable AI models tailored for resource-constrained edge computing devices. By bringing AI closer to the data source, this integration minimises latency and enables more responsive and intelligent IoT environments. Intelligent edge computing offers several additional advantages: reduced bandwidth requirements, ability to perform sensor data fusion at the network edge, and accelerated response times for distributed ML algorithms. Ongoing research and development in intelligent edge computing have the potential to transform various industries. Autonomous vehicles, smart cities, smart homes and healthcare are just a few sectors that stand to benefit from intelligent solutions delivered at the network's edge [1], [38].

The exploration of edge computing in this section reveals a complex and rapidly evolving technological landscape. The analysis of performance considerations highlights the potential of edge computing to significantly reduce latency and network usage, as demonstrated by Alam et al. [32] and Caiazza et al. [36]. However, the performance inversion problem identified by Ali-Eldin et al. [33] underscores the need for careful optimisation and load balancing in edge computing environments. The discussion on platform integrations, particularly the work of Malazi et al. [30] and Vano et al. [14], illustrates the ongoing efforts to seamlessly integrate cloud-native technologies like microservices and containerisation into edge computing environments, despite the challenges posed by resource constraints and

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

network unreliability.

The emergence of intelligent edge computing, as explored earlier, represents a convergence of these trends, aiming to leverage AI and ML to address the unique challenges of edge computing environments. This integration of intelligence at the network edge promises more adaptive, efficient, and autonomous systems. However, it also introduces new complexities in terms of model deployment, data management, and system orchestration. In the next section, the discussion will be centred on AI. The role AI techniques, particularly ML and RL, play in realising the full potential of edge computing will be made clear.

2.4 ARTIFICIAL INTELLIGENCE

AI, a term coined in 1956, aims to construct intelligent machines capable of performing human-like tasks [1]. AI has experienced fluctuations, recently re-surging since the 2010s due to advancements in DL. It is an approach to creating machines that carry out activities typically requiring intelligence. AI covers a broad spectrum of behaviours linked to intelligence such as problem-solving and learning.

ML, a branch of AI, is concerned with developing algorithms that can interpret and make predictions based on data. DL, a subset of ML, enables computers to learn and make decisions without explicit programming. DL utilises artificial neural networks, which are structures that consist of multiple layers of interconnected nodes called neurons, to learn from data, leading to substantial progress in various fields. Figure 2.2 shows how these fields in general relate.

Optimising AI for edge computing encompasses a range of approaches to enhance efficiency, performance, and security. The primary areas where the optimisation is applied include data processing, resource management, AI model adaptation, and security and privacy.

Data processing in edge computing involves reducing the dimensionality of data at the network edge. This process, known as dimensionality reduction, aims to extract contextual knowledge while minimising bandwidth needs. By reducing the dimensionality of data, edge computing devices can perform real-time data analysis and decision-making locally, reducing the need for constant communication with centralised remote servers.

© University of Pretoria

Another area of edge computing optimisation is AI model adaption. It entails modifying machine learning techniques for distributed computing and network edge storage systems. The difficulties of scarce processing resources and the requirement for quick decisions at the network edge are addressed by this adaption process. Model compression, quantisation, and pruning are a few of the methods used to shrink ML models without compromising their accuracy.

Security and privacy are paramount in edge computing, given the sensitive nature of data processed at the network edge. Optimisation techniques in this area focus on protecting data from unauthorised access, ensuring the integrity of data, and preserving user privacy. Privacy algorithms, such as Federated Learning, are used to protect data during transmission and in storage [43]. This technique involves multiple edge computing nodes collaborating to train ML models locally and then sharing the trained parameters with a central server, ensuring data integrity. Additionally, access control mechanisms and authentication protocols are implemented to restrict access to authorised users and prevent unauthorised modifications or deletions of data.

Together, these optimisation areas support edge computing system' safe and effective operation. By optimising data processing, resource management, AI model adaptation, and security and privacy, edge computing can effectively handle the increasing demands of real-time decision-making, data analysis, and AI-powered applications at the network edge. In the next section, a general overview of AI for EI is provided.

2.4.1 Overview of AI in edge computing

The future of data processing at the boundary of a network is changing as a result of AI's integration into edge computing environments. EI enables insights and analytics on resource-constrained devices, improving latency, privacy, and bandwidth usage across various domains [3].

By eliminating the requirement for continuous contact with remote cloud servers, EI facilitates quick decision-making. As a result, latency is drastically reduced, ensuring near-instantaneous responses to time-sensitive tasks, such as autonomous vehicle navigation or industrial process control.

EI also optimises bandwidth usage by lowering the amount of data that must be sent to distant servers. Edge computing devices are capable of doing initial evaluation of data and feature extraction, sending only pertinent data to distant data centres for additional processing. This increase in efficiency reduces

© University of Pretoria

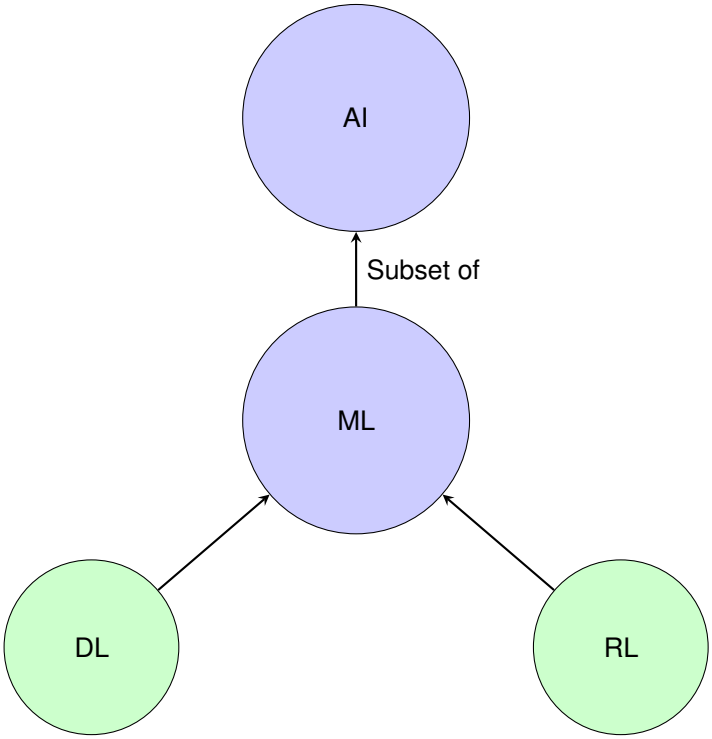

Figure 2.2. A representation to visualise the hierarchical nature of fields within AI.

operating costs, eases network congestion, and permits a larger rollout of IoT devices in situations with limited bandwidth.

The main categories of AI techniques being applied in edge computing contexts, are ML, DL and RL [1]. These categories collectively aim to enhance the efficiency, reliability, and responsiveness of edge computing environments.

2.4.2 Machine learning

Edge ML combines edge computing and ML to provide intelligent solutions near end users, offering the same benefits as EI. Edge ML is a term used to distinguish other applications of AI in edge computing from ML approaches. This is in part because EI faces challenges due to joint constraints from edge computing and AI domains, necessitating efficient solutions in the optimisation areas. To meet the demands of edge ML, researchers are actively developing a range of paradigms and techniques. Two critical areas of focus are edge inference, which aims to optimise the performance of ML tasks, and edge learning, which enables intelligent optimisation of resources for resource-constrained environments [44].

© University of Pretoria

Edge inference aligns with the MSA, enhancing essential attributes like scalability, performance, and decision support at the network's edge. Researchers have delved into innovative approaches to transition monolithic ML applications to microservices within edge computing environments.

ML in MSA (MLMA) stands as a foundational ML architecture that offers design patterns. It aims to transform monolithic ML workflows into microservices, resulting in substantial performance improvements. One notable example is the Scene Classifier application, where MLMA demonstrated a remarkable 67% reduction in processing time compared to the baseline AI model [45]. The applicability of this approach extends beyond image classification, as it can be adapted to enhance a Tourism Recommendation system within a Smart City context.

In the realm of remote surveillance, a real-time feed sensor environment showcased the successful transformation of a monolithic application into a MSA-based ML application. This innovative approach empowers human operators to effectively manage information overload. By reducing event management time by approximately 50%, the redesigned MSA ML application prioritises, selects, and classifies only significant events, enabling operators to focus on critical information [46].

Ortiz et al. [47] further validate the capabilities of ML in handling vast volumes of data for edge inference through their CARED-SOA application. This microservice application delivers real-time, context-aware predictions on air pollutant levels in the vicinity. The system's exceptional performance, processing approximately 300,000 messages with a processing time of under 0.06 seconds per message, exemplifies the potential of ML-powered edge inference.

The synergy between edge computing and MSA creates a fertile ground for cutting-edge applications that leverage the low-latency characteristics of edge computing while maintaining the high throughput of distributed microservices. This convergence creates new opportunities for cutting-edge uses like virtual and augmented reality, enhancing user experiences and enabling innovative use cases across various domains.

In edge inference, the quality and quantity of data play a crucial role in determining the performance of the application. However, real-world datasets often present challenges that can hinder the accuracy and efficiency of ML models. One common issue is missing values. When data points are missing, it can be difficult for ML models to learn the underlying patterns and relationships in the data. This can

© University of Pretoria

lead to biased predictions and reduced accuracy. To address this, various techniques can be employed, such as imputation where missing values are estimated based on the available data.

Another challenge is data imbalance. This occurs when one class or category in the dataset is significantly overrepresented compared to others. This can make it difficult for ML models to learn from the minority class, leading to biased predictions. To mitigate this, techniques such as oversampling or undersampling can be used to balance the distribution of classes in the dataset.

It is crucial to properly handle these data quality challenges in order to guarantee the precision and dependability of edge inference applications. This could entail using data-cleaning methods, like addressing missing numbers, eliminating outliers, and adjusting for biases. It is also important to use appropriate data augmentation techniques to enrich the dataset and improve the generalisation ability of the ML models. By addressing data quality issues, edge inference applications can be developed to deliver accurate and reliable results, ultimately enhancing the user experience and enabling the full potential of real-time inference at the network edge.

Edge learning, on the other hand, is where ML models are utilised to enhance the operation of services in edge computing architecture. ML algorithms are increasingly used to optimise edge computing for microservice applications. RL has emerged as a prominent trend, accounting for 56% of research in this area while microservice applications account for 96% of all the use cases [48]. The other algorithms used are Supervised and Unsupervised Learning algorithms [49].

EI seeks to optimise various network operations, such as application execution and resource allocation to improve overall system performance. The emphasis on RL for this research will be explored in the next section due to its significance in this research.

2.4.3 Reinforcement learning

RL is a type of ML in which an agent gains decision-making skills by acting and getting feedback. An agent's ability to develop a strategy that maximises the total reward over time is the aim. Figure 2.3 shows a depiction of the RL environment. The diagram illustrates the actor-critic architecture within the RL framework. At the core of the RL agent are two key components: the actor and the critic, represented by ellipses. The critic assesses the worth of states and acts, whereas the actor chooses actions according to the present state. These components interact with the environment, depicted as a

© University of Pretoria

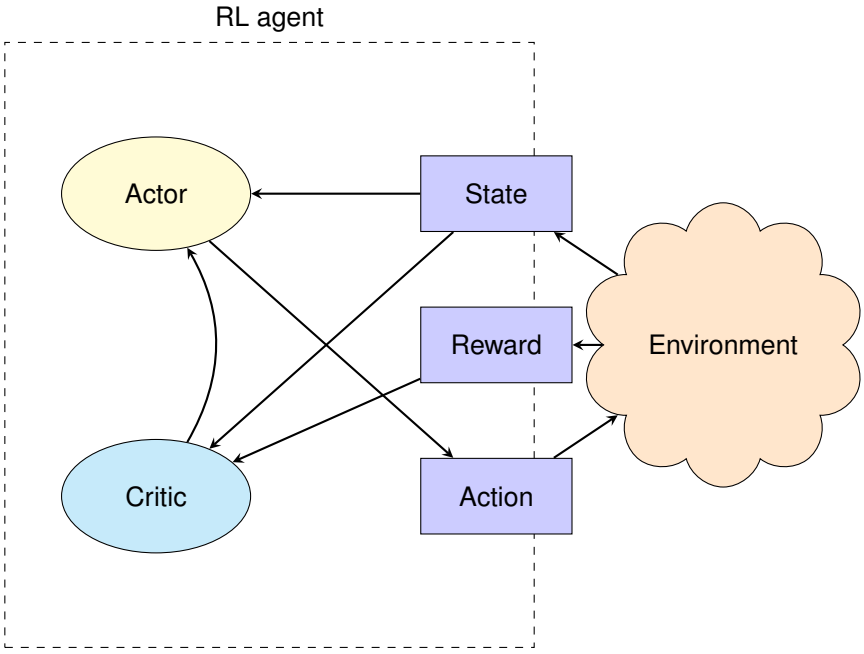

Figure 2.3. A figure that illustrates the actor-critic architecture within an RL framework.

cloud shape, which provides the state and reward feedback. Arrows represent the flow of information: the environment sends a state to both the actor and critic; the actor then selects an action based on this state, which is sent back to the environment. In response, the environment provides a new state and a reward. The critic uses this state and reward information to evaluate the value and generate a response, which is fed back to the actor to improve its policy. Both the actor and critic undergo continuous self-improvement. This cycle of interaction, evaluation, and improvement forms the core of the actor-critic method in RL, enabling the agent to learn and optimise its behaviour over time in complex environments.

RL is considered a method to optimise edge computing networks by enabling adaptive decision-making in dynamic environments. RL can help manage complex network information and improve maintenance and management of edge computing systems such as service placement, task offloading, resource utilisation and reducing latency. This section presents a comparative analysis and review of recent studies that apply RL in broad context ML algorithms to microservices in intelligent edge computing.

Researchers have developed innovative strategies to address the intricate challenges of microservice orchestration. One of these strategies is deep RL algorithms to tackle the multi-faceted nature of

© University of Pretoria

the problems during microservice deployment. Lv et al. [50] introduce a reward sharing deep Q learning (RSDQL) algorithm for multi-objective microservice deployment. RSDQL employs a deep neural network to approximate the value function and incorporates a reward-sharing mechanism to balance multiple objectives, such as resource utilisation, response time, and cost. By leveraging RSDQL, microservices can be deployed in a way that optimises these objectives simultaneously. Specifically, RSDQL achieves a 30% reduction in communication overhead and maintains better load balance across edge nodes during testing. Chen et al. [51] introduced a multiple buffer deep deterministic policy gradient (MB-DDPG) approach that significantly reduces IoT device waiting times by dynamically leveraging heterogeneous edge computing. Their experimental results demonstrated compelling performance improvement, achieving a 32% reduction in average waiting time compared to genetic algorithms and a 44% improvement over random deployment strategies. Hu et al. [52] and Peng et al. [53] both tackled the challenge of optimizing microservice deployment and request routing within edge computing environments to enhance the performance of large-scale applications. Both research teams recognized the critical impact these decisions have on overall system performance. While Hu et al. framed the problem as a multi-objective optimization problem considering deployment, energy consumption, and load balancing, Peng et al. focused on the joint optimization of microservice placement and request routing to alleviate server pressure. Hu et al. introduced a novel reinforcement learning algorithm called reward shaping proximal policy optimization, employing reward shaping techniques for effective and efficient learning. Peng et al., on the other hand, developed the reward memory shaping deep deterministic policy gradient (RMS-DDPG) algorithm, incorporating long short-term memory to enhance learning by prioritizing beneficial actions. Both teams evaluated their algorithms using simulations with both real-world and synthetic datasets. Hu et al.'s algorithm achieved significant improvements in delay, load balancing, and energy consumption compared to baseline algorithms. However, the first fit decreasing algorithm outperformed their solution in terms of deployments, albeit at the expense of resources. Similarly, Peng et al. demonstrated that RMS-DDPG outperformed existing algorithms like MB-DDPG, RSDQL, and other heuristic algorithms. RMS-DDPG achieved comparable improvements in delay and load balancing to Hu et al.'s approach and exhibited superior stability and consistency in achieving rewards compared to other RL algorithms. However, it did not account for user mobility scenarios within the network.

The computational complexity of microservice deployment has prompted researchers to develop advanced representation learning techniques. Tong et al.'s [54] research by leveraging spectral graph theory, graph convolutional networks, and multi-agent reinforcement learning, servers can independ-

© University of Pretoria

ently and collaboratively determine auto scaling strategies. This method effectively reduces the state space for faster processing, minimizing average waiting time, reducing service level agreement violations, and stabilising scaling while considering server heterogeneity. Their proposed algorithm outperforms other algorithms in terms of waiting time and percentile latency, demonstrating its ability to meet service standards and deliver quick response times. Similarly, Wang et al. [55] focus on improving decision-making efficiency in microservice deployment with high dimensional state spaces. Their two-part system incorporates attention mechanisms into reinforcement learning to capture the complex dependencies between microservices, considering factors like topology, size, and type. This results in significant system cost reductions ranging from 20% - 140%. Complementing this, Chen et al. [56] on the other hand combined graph neural networks with deep reinforcement learning, introducing a method that captures intricate microservice structural features and analyses spatial dynamics of edge resource distribution. This approach achieved impressive system costs, improving performance by 20% - 35% and enhancing resource utilisation by 10% - 30%.

These RL-based approaches demonstrate significant improvements over traditional methods across various performance metrics. Tang et al. [57] who aim to optimise the placement of containers for microservice applications across multiple servers, find that RL outperforms baseline approaches, improving delay by 2.9% on average, achieving a 48.5% average improvement in power consumption, and reducing migration cost by 58.4% on average. Quantitative results from other studies further underscore the potential of RL techniques in this domain. Fu et al.'s [58] Nautilus system, while not explicitly using RL, demonstrates substantial improvements, reducing computational resource usage by 23.9%, network bandwidth usage by 53.4%, and deployment costs by 29.9% on average for AI applications compared to existing approaches for RL microservice applications. Wu et al's [59] three dimension dynamic programming based approach for AI microservice deployment achieves average deployment cost reductions of 1.8%, 67.7%, and 71% compared to heuristic methods respectively. Meanwhile, Hu et al. [52] with their RL algorithm discover waiting time reductions of 32 - 44%, energy consumption reductions up to 19.32%. While Peng et al. [53] affirm this through their own research which discovered service delay improvements between 8.6 - 28.5%, and load balancing performance enhancements from 27.5 - 71%. These metrics underscore the transformative potential of reinforcement learning in edge computing environments.

A common thread among these studies is their focus on optimising latency or delay and resource usage during service scheduling. However, the specific metrics and objectives vary, reflecting the

© University of Pretoria

multifaceted nature of the problem space. For example, Wu et al. [59] uniquely address the robustness of AI microservice deployment to edge computing node failures, while Tong et al. [54] consider service level agreement violation rates and autoscaling stability in their evaluation.

Despite the promising results demonstrated by these studies, several limitations are evident. First, mobility scenarios remain largely unexplored, with several prominent research efforts explicitly acknowledging limitations in modeling user network mobility. Second, although there is a clear movement towards using RL techniques to address complex optimisation problems in edge computing environments. Tang et al. [57], Wang et al. [60], and Lv et al. [50] demonstrate significant improvements in various performance metrics using RL-based approaches. However, these studies also reveal a common challenge: balancing multiple objectives such as latency reduction, resource utilisation, and energy efficiency. Most notably, many approaches demonstrate insufficient complexity in modeling intricate microservice dependency relationships, particularly in large-scale distributed environments. Table 2.2 gives a comparison of these papers. This narrow focus may limit the real-world applicability of the proposed solutions, as practical edge computing environments must contend with a broader range of challenges and constraints.

Despite these limitations, the reviewed research presents a compelling narrative of technological advancement. Reinforcement learning techniques have demonstrated remarkable potential in addressing the challenges of microservice deployment. By integrating sophisticated approaches such as graph neural networks, attention mechanisms, and multi-agent strategies, researchers are progressively developing more nuanced and adaptive solutions for optimising edge computing environments.

The collective research suggests that while RL offers promising solutions for dynamic resource management and service orchestration in edge computing, there is still a need for a more robust, generalised model that can adapt to the heterogeneous nature of edge computing environments. This research aims to address these gaps by integrating a broader range of performance metrics, including reliability and energy consumption, into RL-based optimisation models, thus developing a more robust and adaptable model capable of handling the dynamic nature of edge computing environments.

2.5 CONCLUSION

This comprehensive review of microservices, edge computing, and AI reveals the intricate interplay between these technologies in shaping the future of distributed computing systems. Microservices

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 2.2. A table comparing key studies in RL for microservices scheduling.

Consideration	**Chen et al.** [51]	**Yu et al.** [61]	**Tong et al.** [54]	**Wang et al.** [55]	**Peng et al.** [53]	**Hu et al.** [52]	**Chen et al.** [56]
Mobility Scenarios							
Microservice Dependencies			✓				✓
Energy Consumption		✓				✓	
Load Balancing			✓		✓	✓	✓
Service Response Time	✓	✓	✓	✓	✓	✓	✓
Network Resource Usage				✓			
Scalability							✓
Reliability							
Edge Node Heterogeneity	✓			✓			✓

© University of Pretoria

architecture, with its emphasis on modularity and scalability, provides a flexible foundation for deploying applications in edge computing environments. However, as this review shows, the resource constraints and heterogeneity of edge computing devices present significant challenges for microservice deployment and management.

Edge computing emerges as a paradigm shift, bringing computation and data storage closer to the point of need. The analysis of platform integrations and performance considerations highlights the potential benefits of this approach; reduced latency, improved bandwidth utilisation, enhanced privacy, and the complexities involved in its implementation. The notion of intelligent edge computing is a development of edge computing, incorporating AI functionalities to tackle the ever-changing nature of edge computing environments.

The exploration of AI and its techniques, particularly ML and RL, demonstrates their potential to optimise resource allocation, improve decision-making, and enhance overall system performance in edge computing environments. However, the review also underscores the challenges in adapting these AI models to the resource-constrained and distributed nature of the network edge.

Several key research gaps emerge from this review. First, there is a need for more adaptive and context-aware AI models that can efficiently operate in heterogeneous edge computing environments. Second, the integration of microservices with edge computing and AI requires further investigation, particularly in terms of optimising service orchestration and resource management. Third, while significant progress has been made in reducing latency and improving efficiency, there remains a need for more holistic approaches that balance multiple performance metrics, including energy consumption and reliability.

As the field of intelligent edge computing continues to evolve, future research could focus on developing more robust, scalable, and energy-efficient solutions that can seamlessly integrate MSA, edge computing, and AI technologies. This integration holds the promise of enabling more responsive, intelligent, and autonomous systems capable of meeting the growing demands of IoT, smart cities, and other emerging applications. Intelligent edge computing will likely play an increasingly central role in shaping the future of distributed systems and edge computing architectures.

© University of Pretoria

CHAPTER 3 METHODS

3.1 CHAPTER OVERVIEW

This chapter presents a comprehensive methodology for evaluating the proposed BAMPP algorithm for microservice scheduling in EI environments. The chapter is organised into several sections; Section 3.2, Section 3.3, Section 3.4, Section 3.5, Section 3.6, and Section 3.7.

The sections of this chapter, each addresses key aspects of the methodology used to evaluate the proposed BAMPP algorithm for microservice scheduling in edge intelligence (EI) environments. Section 3.2 discusses the foundational design considerations, including the selection of the simulator, datasets, application models, and benchmark algorithms, which underpin the research. Section 3.3 identifies the limitations of this methodology, providing a basis for critical evaluation of its robustness and the broader applicability of the results. Section 3.4 delves into the EdgeAISim simulator, highlighting its suitability for modeling complex edge computing scenarios, the modifications made to tailor it for this study, and its integration with third-party tools. Following this, Section 3.5 details the application models chosen to represent diverse EI scenarios, such as cardiovascular health monitoring, drone coordination swarms, and e-commerce applications. Section 3.6 elaborates on the performance measures used to assess BAMPP's effectiveness, while Section 3.7 presents the benchmark algorithms used for comparative analysis, highlighting their advantages and limitations.

Together, these sections provide a comprehensive framework for evaluating the BAMPP algorithm. By addressing diverse edge computing scenarios and emphasizing both theoretical and practical aspects, the chapter ensures a robust analysis of the algorithm's ability to optimize key QoS factors under resource constraints. This structured approach also facilitates clear insights into BAMPP's performance in comparison to existing state-of-the-art scheduling techniques, establishing its relevance in advancing microservice scheduling in EI environments.

© University of Pretoria

3.2 DESIGN CONSIDERATIONS

This section outlines the key design considerations that shaped the research methodology. The design approach was crafted to address the identified gap in microservice scheduling for EI environments. This section encompasses the selection of an appropriate simulation tool, the use of real-world datasets, the development of diverse application models, and the choice of comparative baseline algorithms. These elements were strategically chosen to ensure a comprehensive evaluation of the BAMPP algorithm, focusing on its ability to optimize QoS factors while adhering to resource constraints in intelligent edge computing scenarios.

The key elements of the research design include:

1. Simulator - The EdgeAISim simulator was selected as the primary research tool due to its ability to model the complexities of real-world microservice deployments in EI. This simulator allows for rigorous testing and refinement of the proposed algorithm under diverse conditions, without the costs and risks associated with real-world implementation.
2. Dataset - The Melbourne CBD dataset, which provides detailed information on network layout and user mobility, was used to create a realistic and dynamic simulation environment. This dataset is particularly relevant as it captures the real-world fluctuations in network traffic and user behaviour that can significantly impact the performance of EI systems.
3. Application models - Three application models were considered in the simulation, each with a different number of microservice dependencies. They are the CHM application, the DRN application and the ECOM application. The CHM application represents a critical use case with stringent QoS requirements, particularly in terms of execution time and fault tolerance. Its inclusion demonstrates the algorithm's capability to handle life-critical applications. The DRN application involves real-time processing and coordination of drones highlighting the algorithm's effectiveness in managing latency-sensitive tasks ensuring timely data processing and migration. The ECOM application is a common, resource-intensive application. It tests the algorithm's efficiency in handling high traffic and variable workloads, emphasizing energy consumption and resource optimization.
4. Algorithms - The proposed algorithm, BAMPP is tested against recognized benchmarks to validate its credibility. The benchmark algorithms considered are; ILP, A2C, and PPO. ILP is a sophisticated approach, offering optimal solutions but often with higher computational costs,

© University of Pretoria

A2C and PPO which are RL-based algorithms, representing advanced, adaptive scheduling techniques.

Each component mentioned; simulator, dataset, application models, metrics, and algorithm directly supports the research objective by providing a comprehensive framework for evaluating the BAMPP algorithm. Subsequent sections elaborate on their significance to addressing the research objectives.

3.3 LIMITATIONS

While this research aims to provide a comprehensive approach to microservice scheduling in intelligent edge computing environments, it is necessary to acknowledge its limitations:

1. Simulation environment: Although EdgeAISim provides a sophisticated simulation of edge computing scenarios, it may not capture all the complexities and unpredictabilites of real-world deployments.
2. Dataset Scope: While the real-world dataset used in this research is representative of edge computing environment, it may not encompass all possible use cases or edge cases that could occur in diverse IoT applications.
3. Performance metrics: This research focuses on optimising latency, bandwidth utilisation, energy consumption and reliability. While these are crucial metrics, there may be other performance indicators relevant in specific edge computing scenarios that are not addressed in this research.
4. Scalability: The findings of this research are based on simulations of specific scales. The performance and efficacy of the proposed BAMPP algorithm in extremely large-scale deployments may require further investigation.
5. Hardware diversity: The research is conducted with a specific set of simulated edge computing devices. The generalisability of the results to all types of edge hardware configurations may be limited.
6. Time frame: The research is constrained by the duration of simulations and the time frame of the dataset used. Long-term effects or rare events that might occur over extended periods may not be captured. For each application, ten simulations were carried out.

© University of Pretoria

Table 3.1. A table comparing EdgeAISim with other simulators.

Feature	EdgeAISim	EdgeSimPy	iFogSim
Task Scheduling	Yes	Yes	Yes
User mobility event	Yes	Yes	Yes
Built-in performance metrics	Limited	Limited	Extensive
RL-based simulation models	Yes	No	No
Extensibility	High	High	Limited
Microservice dependency modelling	Limited	Limited	Limited

These limitations provide opportunities for future research and should be considered when interpreting and applying the results of this study.

3.4 SIMULATOR

This section explores the EdgeAISim simulator, a specialized Python-based tool tailored for modeling AI-driven microservice scheduling in edge computing environments. It examines the simulator's architecture, highlights its unique capabilities, and justifies its selection for this research. Additionally, the section discusses key modifications made to enhance its compatibility with the study's requirements, including the integration of third-party software like OpenAI Gymnasium and Stable-Baselines. These adjustments address simulation gaps such as handling microservice dependencies and incorporating performance metrics. A comparative analysis with other simulators and a detailed account of challenges and solutions in extending EdgeAISim are also presented.

3.4.1 EdgeAISim

EdgeAISim, a Python-based toolkit, is designed to simulate and model AI algorithms in intelligent edge computing environments. It focuses on optimizing power consumption and addressing the challenges of resource management in such environments. Unlike traditional simulators like EdgeSimPy or IFogSim, EdgeAISim incorporates advanced AI-based simulation models for tasks such as task scheduling and mobility support. Table 3.1 compares these simulators side-by-side.

The key components of EdgeAISim, as shown in Figure 3.1 are:

© University of Pretoria

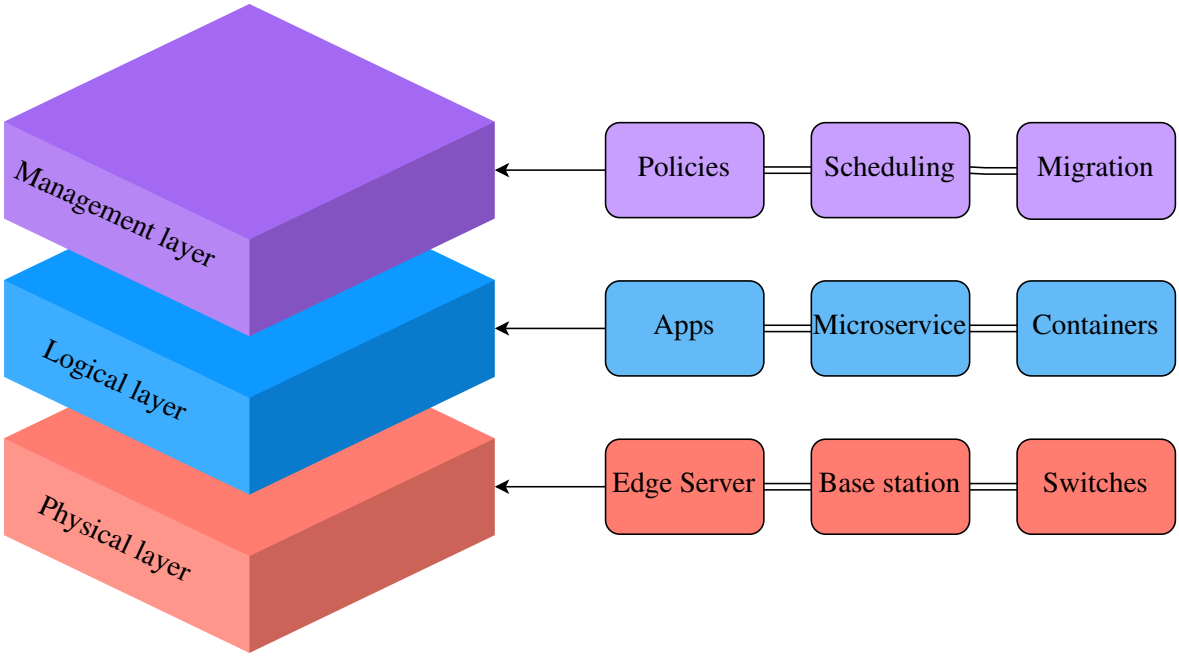

Figure 3.1. A visual representation of the EdgeAISim architecture.

1. Physical layer - In this layer, users and resources, such as base stations, network switches, and edge servers, are abstracted. Users posses end devices like a mobile device which accesses services hosted on edge servers while adhering to a specific mobility model. Base stations within their coverage area offer network connectivity to mobile devices. Network switches serve as intermediaries between base stations and gateway server. Edge servers are responsible for hosting services.
2. Logical layer - In this layer, applications, microservices, and container-based virtualization are abstracted. Each microservice is assigned certain properties, which include CPU demand, memory demand, image, device designation, and availability status.
3. Management layer - This layer is responsible for managing resource policies, encompassing service migration, maintenance activities, and scheduling network flows.

EdgeAISim leverages AI techniques like actor-critic networks to provide an optimization solution for edge computing. However, its primary focus on power minimization limits its usability. While some researchers have adopted it to demonstrate their algorithms' feasibility in intelligent edge computing [62]. EdgeAISim lacks built-in features to address various performance metrics of edge environments. This omission hinders efforts to expand framework's capabilities. Notably, it doesn't include features for microservices dependencies or crucial metrics like energy consumption.

© University of Pretoria

To address this, modifications were made to facilitate simulating microservice dependencies and performance metrics. First by extending the simulation environment with third-party software to create dependencies among microservices. Second, implementing performance metrics based on industry-rated formulae to measure and track edge servers' resources and application runtime performance.

These modifications came with challenges such as prolonged simulation runtime due to the transition from EdgeAISim's code structure to the third-party software's code. Second major challenge was the complexity of integrating new modules with EdgeAISim's existing structure. This was resolved by prioritizing EdgeAISim's components while ensuring consistency with the third-party software's interfacing rules.

To validate these extensions, paper replications were conducted in the new environment on studies that explored microservice scheduling in edge computing without this simulator. The results aligned with previous findings.

3.4.2 Third-party software

In order to extend EdgeAISim to cover the deployment scenario two packages were considered; OpenAI's Gynmasium and Stable-Baselines. Theses two are Python-code packages deigned specifically for RL.

OpenAI's Gymnasium package, also known as OpenAI Gym, is an open-source toolbox designed for implementing RL algorithms and environments. Introduced in 2016, it adheres to a simple and consistent interface, making it accessible to developers of all levels. The purpose of OpenAI Gym is to offer a standardized environment for experimenting with RL algorithms. It simplifies the process of developing and testing agents in diverse environments, facilitating the creation and evaluation of RL models. The core components of OpenAI Gym include:

1. Environment - The environment component served as the core module for creating RL environments. It encapsulates an environment with underlying dynamics that align with the fundamental structure of every RL model. An environment inherits essential functions like step, reset, render, and close. The step function updates the environment based on actions, providing the agent with feedback and additional information. In the microservice scheduling environment for every step

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the agent took, the agent observed the user's location and the edge nodes in proximity to that location, the resource availability of each edge node, the resource demand of each microservice, and then decided which microservice to schedule and to which edge server. The reset function resets the environment to its initial state, giving the agent its first observation input and receiving feedback. The render function visualizes the agent's perspective across various modes. While the close function properly closes the environment. These functions collectively enabled effectively modeling RL by managing the environment's dynamics and providing the agent with the necessary feedback and inputs.

2. Spaces - In the context of OpenAI Gym, the Spaces module encompasses a set of possible observations or states that an agent can perceive within an environment, as well as the set of actions the agent can take. Their purpose is to define the format of valid actions and observations, thereby enabling structured data interaction between the agent and its environment. Observational spaces can be continuous, such as sensor readings, or discrete, like positions on a chessboard. This distinction allows for a comprehensive representation of the environment and the agent's interactions with it. The state space S for the intelligent edge environment, comprised observationas at each time step s_t where $t \in N$ and $N = 0,1,2,\ldots,t$. The system observed resources at each time steip to determine whether constraints are met while pursuing the objective. For example, the state-value or observation of a microservice placed on a node was formulated as (3.1) below.

$$s(i) = \begin{cases} G(x,y) - a\mu s(i,j), \\ \text{when } a = 1. \\ G(x,y) - (a\mu s(i,j) + \sum_{k=0}^{\infty} \mu s(i+k, j+k)), \\ \text{when } a = 0. \end{cases} \tag{3.1}$$

$G(x,y)$ represents the device resource, with x representing CPU resources, y representing Memory resources, a indicates whether migration is initiated, and $\mu s(i,j)$ denotes microservice resources. Specifically, termination occurs as soon as there are no more microservices to be placed, signifying the completion of the learning phase.

In each time step, the agent selected an action from the action space $A = a_0, a_1, \ldots, a_n$ to influence state transitions. The action space was made up of action values a_i. The action state corresponds to the state space, ensuring that each observation in the state space was followed by an action

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

from the action space according to the proposed algorithm. Over time, the environment changed as edge devices' memory or CPU resources were utilized, leading to increased device usage with each placement.

3. Reward system - In RL, the reward system, or reward function is a key factor in determining how the agent behaves. This function delivers feedback to the agent based on its actions. By providing this feedback, the reward function guides the agent towards accomplishing its goals. Crucially, RL algorithms are designed to maximize the cumulative rewards over time on a discount factor $0 < \gamma < 1$, enabling them to learn optimal strategies in complex and dynamic environments. The reward function R_t, designed to minimize computational cost, failure requests, and network usage by maximizing rewards, is described in [63] as (3.2) below.

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-1} r_T. \tag{3.2}$$

The reward value was formulated as (3.3).

$$r = \begin{cases} 1, & \text{when constraints are met.} \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

When the agent placed a microservice successfully, it received a reward of 1; otherwise, it got 0. The goal was to maximise the expected return or accumulated reward.

The OpenAI Gym interface is handicapped when it comes to aspects such as multi-agent environments, iterative tasks with varying lengths and agent partial observability. Regardless, it enabled the training and deployment of the proposed algorithm with efficiency.

Stable-Baselines, an open-source Python library, offers trustworthy implementations of DRL algorithms. These implementations are thoroughly tested against reference codebases and pass all validation tests, ensuring their reliability. The main objective of Stable-Baselines is to provide robust and well-tested implementations of model-free RL methods. The state-of-the-art algorithms; PPO and A2C were provided by Stable-Baselines. Although Stable-Baselines has similar limitations as OpenAI Gym, its features and benefits surpass the limitations by enabling benchmark testing of algorithms against the proposed algorithm.

3.4.3 Dataset

The Melbourne central business district (CBD) dataset [64], sometimes referred to as the EUA Dataset is a publicly accessible dataset containing the detailed location information such as latitude, longitude,

© University of Pretoria

licensing area, postcode, and elevation for edge nodes deployed across the CBD of Melbourne, Australia. The Australian Communications and Media Authority is in charge of overseeing this dataset. The dataset is used to simulate diverse mobility-driven events. The dataset is further customised by segmenting the CBD into multiple blocks, with each block having an edge server at its center and gateways acting as tier-2 nodes. This hierarchical structure helped in creating a realistic simulation environment for intelligent edge computing. The dataset covers the period from January 2019 to April 2021. The Australian Government issued this dataset to assess pedestrian activity before and after COVID-19.

In simulating user mobility patterns within the environment, the Melbourne CBD dataset's location data was essential. The dataset's random mobility models of users served to simulate user mobility during application execution which required microservice migration across edge servers. Additionally, location information from the edge nodes was extracted and used to simulate the network topology and environment for the user. This enabled the simulation of mobility-driven events and varying resource demands as the user traversed the CBD's blocks.

For edge computing researchers, the Melbourne CBD dataset is an invaluable resource. It provides insights into the unique challenges and opportunities of edge computing in a dense urban environment. The dataset also includes a variety of user mobility patterns in addition to the network topology. This data is used to study a wide range of edge computing topics, such as resource allocation, task scheduling, and network optimization.

The Melbourne CBD dataset is particularly valuable because it is representative of a real-world edge computing environment. The data was collected from a variety of edge devices, including base stations, proxy servers, and gateway devices. It provides a realistic view of the challenges and opportunities of edge computing in a dense urban environment. Researchers have used the Melbourne CBD dataset to study a variety of edge computing topics. For example, researchers have used the dataset to study the impact of edge computing on network latency, the energy consumption of edge devices, and to develop new edge computing algorithms and architectures [64].

There are some potential biases of the dataset which must be highlighted. For example, since the original intention of the dataset was for urban planning and management, the dataset's utilization beyond this context deviates from its intended purpose. This is evident from the lack of contextual

© University of Pretoria

information such as weather conditions or special geographic information about the location that could influence the placement of edge nodes or pedestrian behaviour. Also the dataset is limited to a particular period and does not encompass recent developments or long-term patterns that may have transpired beyond that time frame. It fails to provide a comprehensive representation of the area's potential future state and historical evolution.

The specific characteristics of the Melbourne CBD dataset may limit the generalizability of our findings to other urban environments. Cities with significantly different population densities, geographical features, or technological infrastructures may present different challenges for edge computing. To address this, we have plan for future work to test the algorithm in diverse urban settings.

Despite these limitations, the Melbourne CBD dataset was a valuable resource for understanding pedestrian dynamics and network topology in a dense urban environment because these limitations did not reflect in the research.

3.5 MICROSERVICE APPLICATIONS

Microservices are designed within the confines of a system's functionality, which is referred to as the bounded context. Each microservice is deployed in a container instance, contributing to an application-directed acyclic graph (DAG) represented as $\langle \mu s_{set}, \mu s_{relation} \rangle$. The DAG comprises a set of distinct microservices, $\mu s_{set} = \mu s_1, \mu s_2, \ldots, \mu s_n$, and relationships between them, $\mu s_{relation}$. The DAG is characterized by vertices representing microservices and edges representing relationships among them. Figure 3.2 illustrates an example of an application's DAG.

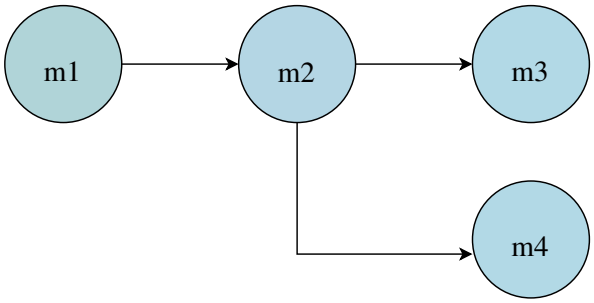

Figure 3.2. A figure illustrating a simple microservices DAG.

In the MSA, when microservice μs_i relies on another microservice μs_j, they are considered related or interdependent, represented as $(\mu s_i, \mu s_j) \in \mu s_{relation}$. The resource characteristics of microservice μs_i can be described using the tuple $\langle m_i, c_i^t \rangle$, where m_i represents the size of the microservice, and c_i^t

© University of Pretoria

signifies the CPU requirement. Additionally, the relationship between $(\mu s_i, \mu s_j)$ incurs a resource cost $\langle o_{ij}, c_{ij} \rangle$, comprising the communication overhead cost o_ij and the CPU length c_{ij}.

To evaluate the effectiveness and versatility of the proposed scheduling algorithm, it was essential to test it across a range of real-world application scenarios. This section details the implementation and characteristics of three distinct microservice applications used in the simulation environment: CHM, DRN, and ECOM. These applications were chosen to represent a spectrum of complexities and requirements, ensuring a comprehensive assessment of the algorithm's performance under diverse conditions.

The CHM application, with its stringent QoS requirements and emphasis on fault tolerance, exemplifies the challenges of deploying critical applications in edge environments. The DRN application highlights the need for real-time processing and efficient resource allocation in latency-sensitive scenarios. Finally, the ECOM application, with its high traffic and variable workloads, stresses the importance of energy efficiency and resource optimization. By testing the proposed algorithm across these diverse applications, this research aims to demonstrate its adaptability and robustness in addressing the multifaceted challenges of microservice scheduling in EI.

3.5.1 Cardiovascular health monitoring application

The CHM application, built to model an IoT-based smart healthcare application, provides the diagnosis of heart disease. It leverages the power of wearable sensors to capture and transmit electrocardiogram (ECG) signals to advanced analysis platforms. This innovative application is comprised of four interconnected microservices, each playing a vital role in delivering real-time health data analytics to patients.

At the core of the application is the Client microservice, deployed on a smartphone. This microservice seamlessly interfaces with ECG sensors, enabling the continuous monitoring and collection of ECG signals. The collected data is then securely transmitted to the Preprocessing microservice.

The Preprocessing microservice, the second microservice, takes the raw ECG signals and applies sophisticated filtering techniques to remove noise. This process ensures that the data fed into the subsequent analysis stages is clean and reliable. The microservice also performs data standardization to ensure compatibility across different devices and sensors.

© University of Pretoria

The Emergency Diagnosis microservice, the third microservice, is responsible for conducting real-time analysis of the ECG signals to detect urgent health conditions. It employs advanced algorithms and ML techniques to identify potential cardiac emergencies, such as arrhythmia, heart attacks, and other life-threatening events. If an urgent condition is detected, the microservice triggers immediate notifications to the patient and healthcare providers, facilitating prompt medical intervention.

The Prediction microservice, the fourth microservice, utilises ML algorithms to analyse historical ECG data and predict future health risks. It leverages advanced statistical models and AI techniques to identify patterns and correlations in the data that may indicate an increased risk of developing cardiovascular diseases. This microservice empowers patients with valuable insights into their heart health, enabling them to make informed decisions regarding lifestyle changes and preventive measures.

The CHM application's modular architecture allows for scalability, flexibility, and continuous improvement. Each microservice can be independently updated and enhanced, ensuring that the application remains at the forefront of technological advancements in healthcare. This architecture would also facilitate the integration of additional features and functionalities in the future, such as remote patient monitoring, personalised health recommendations, and integration with electronic health records.

Overall, the CHM application empowers patients with real-time insights into their heart health, enabling proactive management of cardiac conditions and the prevention of life-threatening events.

3.5.2 Drone coordination swarm application

The DRN application is designed to model a cutting-edge service specifically designed to orchestrate the intricate movements of a swarm of programmable drones. These drones possess remarkable capabilities, including image recognition and obstacle avoidance, enabling them to carry out complex tasks with precision and efficiency.

At the core of this application lies a robust architecture that leverages the power of edge servers. These servers act as the primary computational hubs, handling the majority of the processing required for the drones' operations. By placing the computational power close to the drones, the application minimizes latency and ensures real-time decision-making, which is crucial for maintaining the swarm's cohesion and responsiveness.

© University of Pretoria

With the help of sophisticated software and algorithms, the edge servers can analyse data from the cameras and LIDAR on the drones. This data provides valuable insights into the surrounding environment, allowing the drones to navigate obstacles, avoid collisions, and maintain their desired formation. The edge servers also coordinate the drones' movements, ensuring that they work seamlessly as a collective unit.

While the edge servers handle the majority of the computational workload, the application also incorporates cloud computing when additional resources are needed. This hybrid approach provides scalability and flexibility, enabling the application to handle large-scale swarm deployments or computationally intensive tasks. The cloud serves as a backup and overflow resource, ensuring that the application can meet the demands of even the most complex scenarios.

The microservice application consists of the following microservices:

1. Client microservice – The client microservice is responsible for receiving data from various sensors and transmitting it to other microservices within the system. It acts as a central hub for data collection and distribution.
2. Communication microservice – The communication microservice manages all communication channels between the drone and other microservices. It establishes and maintains connections, facilitating data exchange and coordination among different system components.
3. Navigation microservice – The navigation microservice utilizes sensor data to determine the drone's position, velocity, and flight path. It integrates this information to optimize route planning and trajectory adjustments.
4. Telemetry microservice – The telemetry microservice continually transmits crucial flight data from the drone to the client microservice. This data includes battery level, health status, and current flight conditions, enabling remote monitoring and diagnostics.
5. Mission Planning microservice – The mission planning microservice generates flight plans based on user input and sensor data. It considers factors such as terrain, weather conditions, and obstacles to create safe and efficient flight routes.
6. Collision Avoidance microservice – The collision avoidance microservice leverages sensor data to identify and avoid obstacles in the drone's flight path. It constantly scans the environment, detects potential hazards, and adjusts the drone's trajectory to prevent collisions.

© University of Pretoria

7. Battery Management microservice – The battery management microservice monitors the drone's battery level and provides notifications when the battery is low or needs servicing. It plays a vital role in ensuring uninterrupted flight operations.
8. Failover microservice – The failover microservice acts as a backup system in case the primary navigation or communication systems encounter issues. It seamlessly takes control of the drone, ensuring continuous flight and data transmission.
9. Weather microservice – The weather microservice collects and distributes weather information to other microservices within the system. It provides real-time updates on wind speed, direction, cloud cover, and other weather conditions, enabling the drone to make informed decisions during flight.
10. Authentication and Security microservice – Safeguarding the system against unwanted access is the responsibility of the authentication and security microservice. It authenticates users, verifies their credentials, and implements security measures to safeguard sensitive data and communication channels.
11. Analytics microservice – The analytics microservice collects and analyzes data from the drone's sensors to provide insights into the drone's performance and the environment it is operating in. By utilising this data, one can enhance the safety and effectiveness of the drone, spot trends, and plan the best flight routes.
12. Data Synchronization microservice – The data synchronization microservice ensures that data is kept up-to-date and consistent across all of the microservices in the drone coordination system. This is essential for ensuring that all microservices have access to the most current information and can make informed decisions.
13. Health Monitoring microservice – The health monitoring microservice continuously monitors the health of the drone's systems, including its sensors, actuators, and power supply. This microservice can detect potential problems early on, before they cause the drone to malfunction or crash.
14. Localization microservice – The localization microservice uses GPS and other sensors to determine the drone's position in space. This information is essential for navigation, collision avoidance, and other safety-critical functions.
15. Resource Allocation microservice – The resource allocation microservice allocates resources to the drone, such as bandwidth and computing power. This microservice ensures that the drone has the resources it needs to perform its tasks effectively and efficiently.

© University of Pretoria

16. Traffic Control microservice – The traffic control microservice manages the traffic of drones in a shared airspace. This microservice prevents collisions and ensures that drones are able to operate safely and efficiently.
17. Image Processing microservice – The image processing microservice processes images from the drone's cameras to extract features, such as objects, edges, and textures. This information can be used for object detection, mapping, and other computer vision tasks.
18. Sensor Integration microservice – The sensor integration microservice integrates data from different sensors to create a more complete picture of the environment. This information can be used to improve the drone's navigation, obstacle avoidance, and other safety-critical functions.
19. Logistics microservice – The logistics microservice handles the logistics of drone operations, such as scheduling flights, refueling drones, and maintaining drone fleets. This microservice ensures that drones are available when and where they are needed.
20. Reporting microservice – The reporting microservice generates reports on drone operations, such as flight logs, telemetry data, and analytics. This data is useful for monitoring the drone's performance, pinpointing regions in need of development, and adhering to legal requirements.
21. ML Inference microservice – The ML inference microservice performs real-time ML model inference for object detection. This microservice can be used to identify objects in the drone's environment, such as people, vehicles, and buildings. This information can be used to improve the drone's navigation, obstacle avoidance, and other safety-critical functions.

The DRN application is applied in various industries, including infrastructure inspection, search and rescue operations, and environmental monitoring. Its ability to manage a swarm of drones autonomously and efficiently makes it an invaluable tool for professionals seeking to leverage the power of drones for their operations.

3.5.3 E-commerce application

The ECOM application, designed to replicate a clothing retail shop, offers a seamless and engaging online shopping experience. Users browse through a comprehensive inventory of clothing items, categorised and filtered by various attributes such as size, colour, style, and brand in order to make purchases. High-quality product images and detailed descriptions assist users in making informed purchasing decisions.

© University of Pretoria

One of the key features of the application is the ability for users to add items to their shopping cart. The cart is designed to be user-friendly, allowing users to view their selected items, modify quantities, and remove items as needed. Users can also save their carts for later checkout or share them with friends and family.

To enhance the shopping experience, users can create their own accounts. This enables them to save their personal information, shipping addresses, and payment details for future purchases, streamlining the checkout process. Additionally, account holders can access their order history, track their shipments, and manage their wishlists.

The checkout process is secure and efficient. Users can select from various shipping options based on their preferences and delivery requirements. The application supports multiple payment gateways, ensuring users can choose their preferred payment method. Once the payment is processed, users receive invoices for their orders via email.

Furthermore, the application incorporates a recommendation engine that analyzes users' browsing and purchase history to provide personalized product suggestions. This feature improves the shopping experience by showcasing items that match users' tastes and preferences. Users can also create and manage wishlists, allowing them to save items for future reference or share them with others.

To make the shopping experience more rewarding, the application offers a discount display feature. This feature showcases current promotions, discounts, and special offers on various products. Users can easily identify discounted items and take advantage of these savings.

The application consists of the following microservices:

1. API Gateway microservice – The API Gateway microservice acts as the entry point for all API requests. It is responsible for managing and routing incoming requests to the appropriate downstream microservices based on the request's endpoint and other criteria.
2. Client microservice – The Client microservice represents the user interface of the ECOM platform, such as the web application or mobile app. It is responsible for handling user interactions, displaying product listings, facilitating search and filtering, and managing user accounts. The

© University of Pretoria

Client microservice communicates with other microservices to retrieve data, perform actions, and update the user interface accordingly.

3. Authentication and Authorization microservice – The Authentication and Authorization microservice is responsible for managing user authentication and role-based access control. It handles tasks like user registration, login, password reset, and session management. The microservice also determines the user's permissions and access rights based on their role within the system.
4. User Interface microservice – The User Interface microservice is responsible for managing the front-end components of the ECOM platform, such as the layout, branding, navigation, and overall design. It works closely with the Client microservice to ensure a consistent and user-friendly interface across different devices and screen sizes.
5. Content Management microservice – The Content Management microservice is responsible for managing and maintaining the website's content, including banners, blogs, FAQs, and other informational pages. It provides a user-friendly interface for content editors to create, edit, and publish content, as well as manage content categories and tags.
6. Product Catalog microservice – The Product Catalog microservice manages product listings, categories, and details. It stores information about each product, such as its name, description, specifications, images, and availability.
7. Pricing microservice – The Pricing microservice manages product pricing, discounts, and promotions. It stores the current price for each product, applies any applicable discounts or promotions, and calculates the final price for customers.
8. SEO microservice – The SEO microservice is responsible for optimizing product listings and site content for search engines. It analyzes and optimizes product titles, descriptions, and meta tags to improve the website's visibility in Search Engine Results Pages (SERPs).
9. Search and Filtering microservice – The Search and Filtering microservice provides search functionality and filters for products. It enables customers to look for products by name, category, brand, or other characteristics.
10. Inventory Management microservice – The Inventory Management microservice tracks stock levels and product availability. It maintains real-time inventory information for each product, ensuring that customers can only purchase items that are in stock.

© University of Pretoria

11. Shopping Cart microservice – The shopping cart microservice manages user shopping carts and wish lists. The microservice handles shopping cart adjustments, such as when products are added or removed by customers.
12. Recommendation microservice – The recommendation engine microservice suggests products to users based on their behavior and preferences. It uses sophisticated algorithms to analyze customer data and patterns, suggesting relevant products that align with individual preferences.
13. Order Management microservice – The order management microservice handles order creation, tracking, and history. The microservice enables users to filter products by price range, availability, and other criteria, making it easier for customers to find the products they are looking for.
14. Checkout microservice – The checkout microservice handles the checkout process, including address validation and shipping options. It aggregates all relevant information from other microservices such as the shopping cart microservice to calculate the total amount to pay.
15. Payment Gateway microservice – The payment gateway microservice integrates with various payment providers to process payments. When a customer makes a purchase, the payment gateway facilitates the secure transfer of payment information.
16. Payment Processing microservice – The payment processing microservice manages payment transactions and reconciliations. It ensures secure data transmission, adheres to compliance standards, and protects sensitive information.
17. Shipping and Delivery microservice – The shipping and delivery microservice manages shipping options, tracking, and delivery status. It determines the optimal route for delivering orders based on factors like distance, traffic, and delivery time windows.
18. Returns and Refunds microservice – The returns and refunds microservice handles product returns, refunds, and exchanges. It communicates with the order management and inventory microservices to process returns seamlessly.
19. Reviews and Ratings microservice – The reviews and ratings microservice manages product reviews and ratings by users. The microservice handles reviews and rating updates and ensures that customers always see the most up-to-date information.
20. Promotion microservice – The promotion microservice manages marketing campaigns, vouchers, and promotions. It manages multiple promotions and their combinations with priority and stackability.

© University of Pretoria

21. Notification microservice – This microservice is responsible for sending email, SMS, and push notifications to users. It integrates with various communication channels and ensures timely and reliable delivery of notifications.
22. User Behaviour Tracking microservice – This microservice tracks user behaviour on the platform. It collects data on page views, clicks, and interactions, helping in understanding user preferences and behaviour patterns. This information is used for analytics, personalisation, and improving user experience.
23. Tax Calculation microservice – This microservice calculates taxes based on user location and applicable laws. It integrates with tax authorities and ensures accurate tax calculations, reducing the risk of legal and financial issues.
24. Analytics microservice – This microservice collects and analyses data from various sources, including user behaviour, transactions, and marketing campaigns. It provides insights into customer behaviour, sales trends, and overall platform performance.
25. Compliance and Legal microservice – This microservice ensures compliance with legal and regulatory requirements. It monitors changes in laws and regulations, updates policies and procedures, and ensures that the platform operates within legal frameworks.
26. Logging and Monitoring microservice – This microservice logs system activities and monitors performance metrics. It helps in identifying and resolving issues, optimizing performance, and ensuring the stability of the platform.
27. Cache Management microservice – This microservice handles caching to improve performance. It stores frequently accessed data in memory, easing the burden on the database and enhancing response times.
28. Media Management microservice – This microservice manages product images, videos, and other media assets. It ensures that media files are stored securely, optimized for performance, and accessible to users and other services.
29. Load-Balancing microservice – This microservice distributes traffic among various services to ensure reliability and scalability. It monitors the health of services and automatically redirects traffic to available instances, preventing outages and performance issues.
30. Session Management microservice – This microservice manages user sessions and state. It stores user-specific data, such as shopping cart contents and preferences, and ensures that users have a consistent experience across devices and sessions.

© University of Pretoria

31. Message Queue microservice – This microservice handles asynchronous communications between services. It ensures that messages are delivered reliably, even if one of the services is temporarily unavailable.
32. Business Intelligence microservice – This microservice provides advanced analytics and business intelligence reporting. It combines data from various sources to identify trends, patterns, and opportunities, helping businesses make informed decisions.
33. Data Synchronization microservice – This microservice ensures data consistency across different services. It synchronizes data in real-time, preventing inconsistencies and ensuring that all services have access to the most up-to-date information.
34. Customer Support microservice – This microservice manages customer support tickets and live chat. It provides a centralized platform for customers to get assistance, resolve issues, and provide feedback, helping businesses improve customer satisfaction and loyalty.

Overall, the ECOM microservice application models a comprehensive online shopping solution for clothing enthusiasts. Its user-friendly interface, rich features, and personalized shopping experience reflect the needs of modern consumers, ensuring a convenient and enjoyable online shopping journey.

3.5.4 General operational framework

Across the microservice applications: CHM, DRN, and ECOM, the operational cycle comprises of four essential stages: prerprocessing, data filtration, data separation, and aggregation. These stages are tailored to each application's unique requirements but follow a consistent framework for ensuring QoS and efficient resource utilisation.

1. Preprocessing – In this stage raw data is collected from diverse sources for example sensors and user inputs, and undergoes initial formatting, cleaning, and transformation. This step ensures compatibility across microservices, reducing errors in subsequent processes.
2. Data filtration – In this stage irrelevant data is removed using application-specific filters, thus optimising bandwdith and computational resource usage. For example, in latency-sensitive applications, noise and redundant data are eliminated early to avoid performance bottlenecks.

© University of Pretoria

3. Data separation – In this stage data is categorised and routed based on task priority and type. Real-time tasks are prioritised over long-term analytics, ensuring responsiveness under constrained resources.
4. Aggregation – Outputs from the distributed microservices are combined into a cohesive form for decision making or user presentation. Aggregation ensures a unified result despite the inherent distribution of microservices.

In the CHM application, the operational cycle begins with preprocessing, where raw ECG signals from wearable sensors are subjected to filtering techniques such as adaptive noise removal to eliminate baseline wander. This ensures that subsequent analysis stages receive clean, standardised data. Following this, data filtration removes biologically irrelevant signal features, retaining critical patterns like R-peaks and QRS complexes that are essential for diagnostic accuracy. The filtered data is then separated into two distinct processing pipelines: one for real-time emergency detection, which identifies critical conditions and another for historical analysis, where machine learning models predict long-term cardiovascular risks. The outputs from these processes are aggregated inot comprehensive reports, enabling healthcare providers and patients to receive actionable insights through a unified interface. This modular approach ensures responsiveness for time-senstive diagnostics while maintaining the robustness required for predictive analysis.

The DRN application integrates preprocessing, filtration, separation and aggregation to manage complex swarm operations efficiently. Initially, sensor data from drones, such as camera feeds and LIDAR scans, undergoes preprocessing to normalise inputs and align them with the system's analytical frameworks. Noise and redundant telemetry readings, which could otherwise overwhelm the system, are filtered out during the data filtration phase, ensuring that only critical navigational and environmental information is retained. Data separation prioritises real-time processing for tasks such as obstacle detection and trajectory optimisation, while lower-priority tasks, including long-term environmental mapping, are deferred to ensure uninterrupted swarm operations. The aggregation phase combines insights from multiple drones into a cohesive situational model, enabling ground control systems to maintain efficient and coordinated swarm behaviour.

In the ECOM application, the operational cycle emphasises scalability and personalisation to enhance the user shopping experience. During preprocessing, user interaction data, such as browsing histories and search queries, is categorised and anonymised to comply with privacy standards. The data filtration

© University of Pretoria

phase eliminates redundant or invalid records, such as duplicate entries or bot-generated interactions, ensuring the integrity of the analytics pipeline. Data separation allows for the prioritisation of tasks; for example, real-time operations like inventory synchronisation and payment processing take precedence over periodic activities such as personalised recommendation generation. Aggregation consolidates outputs from diverse microservices including product catalogs, recommendation engines, and order management systems into a seamless and responsive interface.

3.6 PERFORMANCE MEASURES

Evaluating the effectiveness of microservice scheduling algorithms in EI requires a comprehensive assessment of various performance metrics. The main performance indicators for the state-of-the-art algorithms and the suggested scheduling method are described in this section. These metrics provide a multifaceted view of the algorithm's behaviour, encompassing aspects related to QoS, network efficiency, and energy consumption.

The QoS metrics, including execution time, migration latency, and reliability, are essential for understanding how the proposed algorithm affects the responsiveness and overall user experience of the applications. The Network usage metric shed light on the algorithm's impact on bandwidth utilization and communication overhead, which are critical considerations in resource-constrained edge environments. Finally, the energy consumption metric provided insights into the algorithm's energy efficiency, which is vital for sustainable and cost-effective EI deployments.

By considering these diverse performance measures, this research aims to provide a holistic evaluation of the proposed algorithm, demonstrating its ability to optimize various performance aspects and outperform existing approaches in the dynamic intelligent edge environment.

3.6.1 Quality of service

The QoS metrics used to assess the performance of the algorithm include execution time, migration latency, and reliability. These metrics provide insights into how well the algorithm handles critical conditions encountered in different application scenarios.

Execution time measured the duration it took for the algorithm to complete its task. This metric is crucial because it directly impacts the responsiveness and efficiency of the system. A longer execution

© University of Pretoria

time results in delayed results or system bottlenecks, particularly in time-sensitive applications. By optimizing the execution time, the algorithm ensures prompt and seamless operation.

Migration latency conveys the amount of time required for the algorithm to migrate from one resource to another. In dynamic environments, where resources are constantly changing or failing, efficient migration is essential to maintain continuity of service. High migration latency can lead to interruptions or delays in processing, affecting the overall user experience. Minimising migration latency enables the algorithm to adapt swiftly to changing conditions, ensuring uninterrupted performance.

Reliability, measured the algorithm's ability to withstand and recover from unexpected failures or errors. In complex systems, faults are inevitable, and the algorithm must be equipped to handle them gracefully. High reliability ensures that the algorithm continues to function correctly even in the face of adversity, preventing data loss or system crashes. By incorporating fault tolerance mechanisms, the algorithm can maintain its integrity and deliver consistent results despite potential disruptions. The reliability mechanism employed in this research is the standard deviation of reward scores. The consistency of each algorithm can be gauged by examining the standard deviation of the algorithm's reward scores. These scores served as a metric for evaluating how effectively the algorithm attained the research objective. A problem with RL methods that doesn't exist with mathematical programming methods is that the empirical returns tend to have a large scale and lead to high variance of the rewards. During deployment, an RL algorithm with a high standard deviation may experience sporadic successes followed by abrupt failures. Conversely, an algorithm with a lower standard deviation exhibits a more stable range of variation in reward scores. This consistency allows for more predictable results, minimizing the likelihood of unexpected failures.

To thoroughly assess the algorithm's performance under various conditions, these QoS metrics play a crucial role. By measuring aspects such as execution time, migration latency, and reliability, insight is gained into the algorithm's efficiency, adaptability, and resilience. It is imperative to optimise these measures in order to guarantee that the algorithm satisfies the demands of various application scenarios. These scenarios include CHM, which emphasizes high computational costs; DRON, which handles dynamic resource allocation; and ECOM, which operates in a financial setting.

© University of Pretoria

3.6.2 Network usage

During the execution of a microservices-based application, the individual microservices communicate with each other by making API calls over the network. This process of making API calls consumes network bandwidth, which is referred to as network usage. The amount of network usage can vary depending on the frequency and size of the API calls, as well as the number of microservices involved in the communication.

Network usage is a critical factor in evaluating the performance of a microservice scheduling algorithm within the intelligent edge framework. This is because efficient network usage can help to optimize bandwidth utilization and reduce latency, which can lead to improved overall application performance.

It is important to use a microservice scheduling algorithm that takes network usage into account when making scheduling decisions. This can help to ensure that microservices are scheduled on hosts that are close to each other, which can reduce the distance that API calls need to travel and improve performance.

The calculation of network usage involved simulating the data transfer activities of microservices under varying user requests and network conditions. Network usage can be expressed mathematically as (3.4) below.

$$U = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{\mu s=1}^{R_i}\frac{P*8}{L_{\mu s}+T_{\mu s}}\right), \tag{3.4}$$

where N is the number of users, R_i is the number of requests made by user i, P is the payload size in bytes, $L_{\mu s}$ is the latency for microservice μs in seconds and $T_{\mu s}$ is the transfer time.

The payload size, transfer time and latency are critical factors in determining the network usage of a microservice. The quantity of data that must be delivered is indicated by the payload size, and it is typically measured in bytes. The transfer time is the time required to transfer the payload given the sampled bandwidth, and it is typically measured in seconds. While, latency is the time at which the microservice takes to be transferred, and it is measured in seconds.

The microservice scheduling algorithm directly impacts the performance and efficiency of network

© University of Pretoria

usage. By evaluating the network usage, insight is gained into the bandwidth utilisation of the algorithm. An efficiently running algorithm avoids bottlenecks during bandwidth consumption.

The network usage of a microservice can be a significant factor in its overall performance. If a microservice consumes too much bandwidth, it can slow down other services that are running on the same network. Therefore, network usage metric is important in assessing the feasibility and effectiveness of the proposed microservice scheduling algorithm within the EI framework.

3.6.3 Energy consumption

Energy consumption is a pivotal factor in determining the effectiveness of the microservice scheduling algorithm within the EI framework. This is because edge devices are often characterized by limited resources, including energy, which can be quickly depleted if not managed efficiently. Therefore, it is imperative that the microservice scheduling algorithm took into account energy consumption when making decisions about which microservices to schedule and where to schedule them.

There are several factors that influence the energy consumption of microservice scheduling algorithms. These include:

1. The number of microservices being scheduled - The more microservices that are scheduled, the more energy will be consumed.
2. The size of the microservices - Larger microservices consume more energy than smaller microservices.
3. The complexity of the microservices - More complex microservices consume more energy than less complex microservices.
4. The scheduling policy used - Different scheduling policies can result in different levels of energy consumption.

To minimize energy consumption, BAMPP took into account these factors. By carefully considering energy consumption, BAMPP improves the sustainability of the EI network.

The calculation of energy consumption involves quantifying the energy consumed by different components; CPU and memory of a device while running the microservices. The total energy usage is

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

then the sum of a base energy consumption and the energy consumed by CPU and memory. It can be expressed mathematically as (3.5) below.

$$E = E_{base} + E_{cpu} + E_{mem}, \tag{3.5}$$

where E_{base} is the base energy consumption, $E_{cpu} = \alpha(\frac{U_{cpu}}{L_{cpu}})$ represents the CPU energy consumption, $E_{mem} = \beta(\frac{U_{mem}}{L_{mem}})$ represents the memory energy consumption, U_{cpu} is the CPU capacity used, L_{CPU} is the CPU usage limit, U_{mem} is the used memory, L_{mem} is the maximum memory usage possible.

The microservice scheduling algorithm directly impacts the sustainability and operational efficiency of energy usage. By evaluating the energy consumption, insight is gained into the sustainability of the algorithm, and the system longevity gained by the algorithm. Once the algorithm proves to meet operational energy-efficient parameters it demonstrates that it can be adapted into real-world settings this demonstrates sustainability. In terms of system longevity, by managing energy consumption effectively, the algorithm reduces wear and tear on hardware components thereby extending their operational lifespan.

The energy usage is a pivotal metric in assessing the feasibility and effectiveness of the proposed microservice scheduling algorithm within the EI framework. By simulating realistic energy consumption patterns and analysing the impact on system performance, we derive meaningful insights into the algorithm's efficiency and its potential for real-world deployment.

3.7 MODELS

In microservice scheduling, the primary objective is to establish a policy that guides decision-making to achieve a desired outcome. This policy ensures continuous success in scheduling microservices on the intelligent edge infrastructure. The scheduling problem is determined to be NP hard, meaning simple heuristics struggle to determine policies that meet all the constraints in polynomial time.

To address this challenge, mathematical programming and ML algorithms, such as RL, have emerged as the most promising alternatives to solve this problem [6]. These algorithms can handle the complexity of the scheduling problem and find optimal or solutions that are almost optimal in fair amount of time.

© University of Pretoria

Mathematical programming approaches involve formulating the scheduling problem as a mathematical optimization model. The objective function of the model represents the desired outcome, such as minimizing latency or maximizing resource utilization. Constraints are then added to the model to ensure that all the requirements of the microservices are met. The model is then solved using optimization techniques to find the best possible solution.

ML algorithms, particularly RL, offer another powerful approach to microservice scheduling. RL involves training an agent to learn the optimal scheduling policy through interaction with the environment. The agent receives rewards or penalties for its scheduling decisions, and over time, it learns to make better decisions that maximize the desired outcome.

The use of mathematical programming and ML algorithms in microservice scheduling has several advantages. First, these algorithms can handle large-scale scheduling problems with a large number of microservices and constraints. Second, they can find optimal or near-optimal solutions in a reasonable amount of time. Third, these algorithms can be adapted to different types of microservices and edge environments.

On this premise, to ensure policy success, there are three approaches considered. One option was integer linear programming (ILP), a mathematical programming model. Alternatively, the advantage actor-critic (A2C) RL algorithm was used, where the agent's initial state is compared to the initial action value to determine the advantage of one over the other. This advantage indicates the best option for maximizing rewards, and by choosing the best states, this policy can be applied for the entire trajectory. A third approach involved optimising the policy using a surrogate objective function and a clipping mechanism, resulting in the best decision for the agent that maximises rewards for the entire trajectory. This is known as the Proximal Policy Optimization (PPO) algorithm.

In contrast to these existing approaches, we propose the BAMPP algorithm, which draws inspiration from both the A2C and PPO algorithms.

3.7.1 Integer linear programming

ILP is a powerful optimization technique used to find the best solution from a set of discrete choices. It entails creating a mathematical model that encapsulates the problem, defining an objective function to be optimized, and specifying a set of constraints that the solution must satisfy. For this research, ILP

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

is employed to address the challenging task of determining the optimal scheduling of microservices across edge devices.

The primary goal of the ILP model is to minimize resource usage while ensuring that all functional and non-functional requirements are met. The objective function quantifies the total resource consumption across all edge devices, considering factors such as CPU utilisation and memory usage requirements. By minimising this objective function, the ILP model aims to find a scheduling strategy that efficiently utilises the available resources.

The objective of the ILP model is to minimise the total resource usage (memory and CPU) across all microservices and devices. This can be expressed mathematically as:

$$Minimize \sum_{\mu s \in M} \sum_{d \in D} x_{\mu sd}(m_{\mu s} + c_{\mu s}), \tag{3.6}$$

where M is the set of all microservices, D is the set of all devices, $x_{\mu sd}$ is a binary decision variable indicating whether microservice μs is deployed on device d. $m_{\mu s}$ is the memory requirement of microservice μs. $c_{\mu s}$ is the CPU requirement of microservice μs.

To ensure the feasibility and correctness of the solution, the ILP model incorporated several constraints. Firstly, it imposed the requirement that each microservice must be deployed at least once, ensuring that all necessary functionalities are provided. Secondly, it specified that each edge device must host at least one microservice, promoting load balancing and preventing resource under-utilization. Additionally, the model included constraints to guarantee that the resource limits (CPU, memory) of each edge device are not exceeded. These constraints help maintain the stability and performance of the deployed microservices.

The constraints are given as:

1. Deployment constraint - This constraint ensures that each microservice is deployed at least once across all the devices in the system. It is formulated by (3.7) below. This constraint is important to ensure that all the microservices are available and accessible to the users.

$$\sum_{d \in D} x_{\mu sd} \geq 1 \quad \forall \mu s \in M. \tag{3.7}$$

2. Device hosting constraint - This constraint states that each device must host at least one microservice. It is formulated by (3.8) below. This constraint is important to ensure that all the

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

devices are utilized efficiently and that there are no idle devices.

$$\sum_{\mu s \in M} x_{\mu sd} \geq 1 \quad \forall d \in D. \tag{3.8}$$

3. Memory constraint - This constraint ensures that the total memory usage on each device must not exceed its memory limit. It is formulated in (3.9) below. This constraint is important to prevent the devices from running out of memory and crashing.

$$\sum_{\mu s \in M} x_{\mu sd} * m_{\mu s} \leq L_{mem} \quad \forall d \in D. \tag{3.9}$$

4. CPU constraint - This constraint ensures that the total CPU usage on each device must not exceed its CPU limit. It is formulated in (3.10) below. This constaint is important to prevent the devices from overheating and failing.

$$\sum_{\mu s \in M} x_{\mu sd} * c_{\mu s} \leq L_{cpu} \quad \forall d \in D. \tag{3.10}$$

5. Dependency constraint - This constraint ensures that microservices with dependencies must be deployed on devices only if their dependencies are also deployed on the same devices. It is formulated in (3.11) below. This constraint is important to ensure that the microservices can communicate with each other and function properly.

$$x_{\mu sd} \leq \sum_{t \in \mu s_relation} x_{td} \quad \forall \mu s \in S, \forall d \in D. \tag{3.11}$$

The decision variables $x_{\mu sd}$ in the ILP model represent the deployment decisions for each microservice. Each decision variable is associated with a binary value, indicating whether the corresponding microservice is deployed on a particular edge device or not. By optimizing these decision variables, the ILP model determines the optimal allocation of microservices to edge devices.

The output of the ILP model consists of the optimal deployment strategy, which specifies the assignment of each microservice to a specific edge device. This deployment strategy minimizes the total resource usage while satisfying all the constraints. The output also includes information about the resource utilisation of each edge device, helping monitor and manage the deployed microservices.

Overall, the ILP-based approach provides a systematic and efficient method for optimizing microservice deployment in edge computing environments. It enables the selection of the best deployment strategy that minimizes resource consumption while ensuring the reliable and efficient operation of the microservices.

© University of Pretoria

3.7.2 Advantage-actor critic

The A2C algorithm is an online policy-gradient algorithm that combines the principles of RL and function approximation. It consists of two main components: an actor (policy) network and a critic (value) network. Both networks are typically implemented as neural networks.

The actor network represents the policy function, which determines the probability distribution of actions given a state. The goal of the actor network in a given state is to optimise the anticipated gain over all feasible actions.

The critic network represents the value function, which estimates the expected long-term reward for a given state and action pair. The critic network helps the actor network focus its updates on actions that lead to better-than-expected outcomes.

During training, the A2C algorithm interacts with the environment by taking actions and observing the resulting rewards and states as shown in Figure 2.3. The actor network is modified to increase the probability of choosing actions that result in higher rewards. The critic network is updated to better estimate the value of each state-action pair.

The A2C algorithm has several advantages over other policy-gradient methods. First, it is an online algorithm, which means it can learn from experience without requiring a large batch of data. Second, it uses function approximation to represent the policy and value functions, which allows it to handle continuous state and action spaces. Finally, the A2C algorithm is relatively easy to implement and train.

The A2C algorithm has been successfully applied to various RL tasks, including robotics, video games, and natural language processing. It is a powerful tool for learning complex policies from high-dimensional observations.

3.7.3 Proximal policy optimisation

The PPO algorithm is a policy gradient RL method intended to enhance the learning process's convergence and stability. It achieves this by maintaining policy updates within a reasonable range, preventing drastic changes that can destabilize the learning process.

© University of Pretoria

At its core, PPO is a variant of the actor-critic method, which involves two components: the actor and the critic. In traditional actor-critic methods, the actor and critic are separate neural networks responsible for learning the policy and value function, respectively. However, in PPO, these two components share a single neural network architecture. This approach simplifies the training process and promotes more efficient learning.

One of the key features of PPO is the use of a clipped objective function. In order to prevent the new policy from departing too much from the previous policy, PPO clips the policy gradient when revising the policy. This clipping mechanism prevents large and potentially harmful updates, which can lead to instability and divergence.

Additionally, PPO introduces the concept of a surrogate objective function, formulated from [65] as (3.12) to reduce variance in the policy gradient estimate. The surrogate function is based on the ratio of new policy probabilities to old policy probabilities. This technique helps improve the efficiency and accuracy of the learning process.

$$L^{CLIP}(\theta) = min(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}A^{adv}(s,a), clip(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}, 1-\varepsilon, 1+\varepsilon)A^{adv}(s,a)), \tag{3.12}$$

where $L^{CLIP}(\theta)$ is the clipped surrogate objective function, $\pi_\theta(a|s)$ represents the probability of choosing action a in state s according to the new policy, $\pi_{\theta_{old}}(a|s)$ denotes the probability of taking action (a) in state (s) according to the old policy, $A^{adv}(s,a)$ is the advantage function, representing the advantage of taking action a in state s over the baseline value, and ε is a hyper-parameter controlling the clipping range.

In essence, PPO stands as an online policy-gradient RL method. It aims to improve stability and convergence of the learning rate by ensuring that policy updates remain within an appropriate range. It combines the actor-critic architecture with a clipped objective function and surrogate objective function to optimise the learning process. By doing so, PPO stands out as an efficient and effective RL method, making it a widely used and successful algorithm in various RL applications.

© University of Pretoria

3.7.4 Bi-generic A2C microservice proxy policy model

The proposed BAMPP algorithm represents a significant advancement in the domain of microservice scheduling within intelligent edge computing environments. It is designed to optimise the scheduling of microservices across edge servers, aiming to enhance QoS factors such as execution time, migration latency, and reliability. By leveraging RL techniques, BAMPP effectively balances computational loads while considering resource constraints and dynamic network conditions. The BAMPP algorithm is outlined in Algorithm 1.

Algorithm 1 BAMPP algorithm.

Input: Edge Servers information, Application information **Output:**Decision Policy

Initialize learning rate a, discount factor γ, clip range ε
Initialize actor network, $\pi(a|s,\theta)$ and critic network $V^{\pi}(s,w)$
for episode $= 1,\ldots,T$ **do**
  for time slot$t = 1,\ldots,t_{\max}$ **do**
    Select action $a_t \sim \pi(\cdot|s_t,\theta)$
    Execute a_t, and observe the reward r_t and the next state s_{t+1}
    Calculate the advantage function:
    $A^{\pi}(s_t,a_t) = r_t + \gamma V(s_{t+1},w) - V(s_t,w)$
    Calculate the ratio of probabilities:

$$\text{ratio} = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

    Update the actor network:

$$\theta \leftarrow \theta_{old} + a\nabla_{\theta}\log\left(\text{ratio}_{clipped}\cdot A^{\pi}(s_t,a_t)\right)$$

    Update the critic network:
    $w \leftarrow w_{old} - a\nabla_{w}\frac{V}{V_{old}}$
  end for
end for

The BAMPP algorithm utilised key parameters including the learning rate, discount factor, and clipping range. actor and critic networks are established to model the policy and value function, respectively. The algorithm trained over a varied number of episodes depending on the application, with each episode simulating a series of time slots. At each time slot, an action was selected based on the current

© University of Pretoria

state and policy. The action was then executed, yielding a reward and the agent transitioned to a new state.

The advantage function was computed to quantify the advantage of a particular course of action in comparison to the anticipated value in a particular state. The ratio of probabilities was then calculated to compare the new policy against the old policy. The actor network parameters were updated using the clipped probability ratio and the advantage function, while the critic network parameters were updated to improve the value function estimation.

The proposed BAMPP algorithm, a cutting-edge RL technique, has proven to be a formidable performer, eclipsing established algorithms like PPO and A2C. Its exceptional capabilities are attributed to several key advancements that set it apart from its predecessors.

Firstly, BAMPP employed a novel approach to policy optimization. Unlike PPO, which relies solely on maximising the policy's probability, BAMPP took a more comprehensive approach. It considered not only the probability but the ratio of the current policy to the former policy and the ratio of the current state-value to the previous state-value in optimising the critic, encouraging exploration and preventing premature convergence. This enabled BAMPP to navigate complex environments effectively, discovering hidden opportunities and uncovering optimal strategies that might otherwise remain elusive.

Furthermore, BAMPP considered the surrogate loss clipping regularisation technique from PPO that clips excessive policy changes. Finding a balance across exploration and exploitation is crucial in RL. While exploration helps discover new opportunities, excessive exploration can lead to unstable policies and suboptimal performance. BAMPP's regularization technique mitigated this issue by discouraging abrupt policy changes, promoting stability and encouraging the algorithm to focus on promising areas of the environment.

These advancements collectively contributed to BAMPP's superior performance. In extensive evaluations across various benchmarks, BAMPP consistently outperformed PPO and A2C, achieving higher returns and demonstrating greater robustness in complex and challenging environments. The BAMPP algorithm represents a significant step forward in RL. BAMPP efficiently balanced the load across edge servers, reducing overall execution time and improving resource utilisation. It minimised

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

migration latency, ensuring seamless operation for latency-sensitive applications. The adaptive nature of BAMPP allowed it to handle dynamic network conditions and resource constraints, thereby enhancing reliability. Furthermore, by taking network circumstances and resource limitations into account, BAMPP improved energy consumption, making it appropriate for contexts with limited resources. The workflow of the algorithm is illustrated in Figure 3.3.

3.8 CONCLUSION

This chapter has outlined a comprehensive methodology for evaluating microservice scheduling algorithms in EI environments. The approach leveraged the EdgeAISim simulator, extended to incorporate microservice dependencies and additional performance metrics beyond power consumption. The Melbourne CBD dataset provided a realistic foundation for simulating dynamic edge computing scenarios, while the three diverse application models (CHM, DRN, and ECOM) represented a range of real-world use cases with varying QoS requirements.

The performance measures of execution time, migration latency, and reliability have been carefully selected to address the key challenges in edge computing environments. These metrics, along with considerations of network usage and energy consumption, provided a multi-faceted evaluation framework for the proposed BAMPP algorithm.

Through a comparative analysis between BAMPP and well-known state-of-the-art algorithms, such as ILP, A2C, and PPO, this methodology seeks to illustrate how well the suggested approach optimises microservice scheduling while respecting resource limitations. The design considerations and methodological choices directly addressed the research objectives by:

1. Enabling the evaluation of QoS factors across multiple applications in a realistic edge computing environment.
2. Providing a framework for assessing the algorithm's performance in managing resource constraints and dynamic network conditions.
3. Allowing for comparative analysis against state-of-the-art scheduling algorithms.

While this methodology provided a robust framework for evaluation, it is critical to recognise its limitations. The reliance on simulation, though necessary for this research, may not capture all

© University of Pretoria

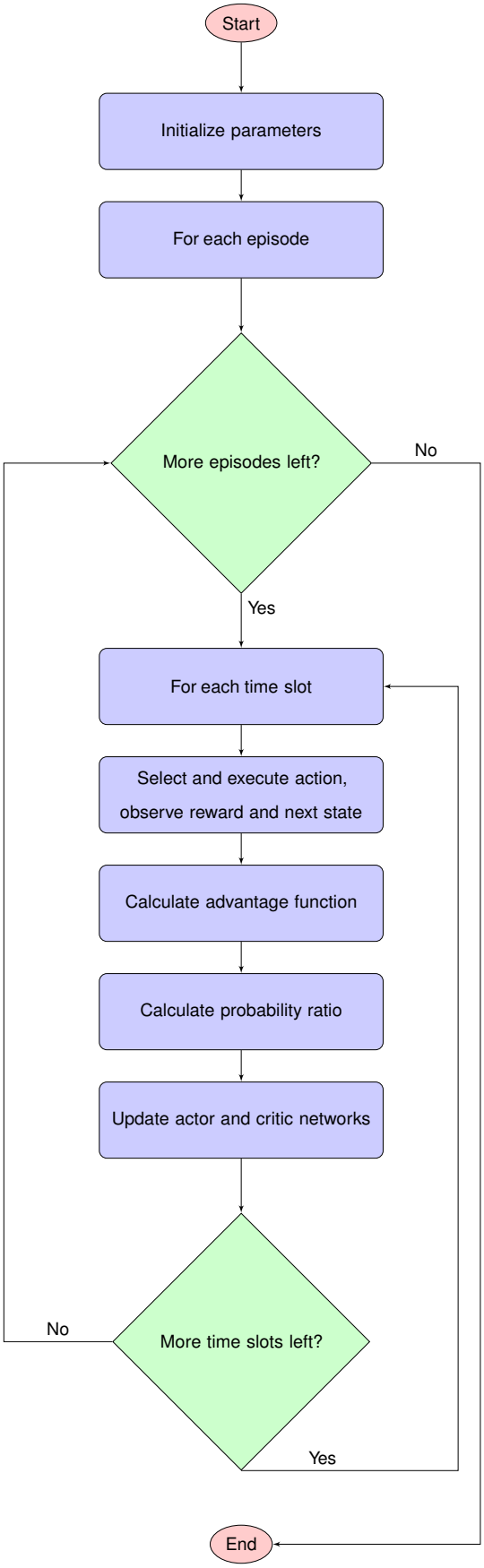

Figure 3.3. A flow diagram that represents the main steps of the BAMPP algorithm.

© University of Pretoria

nuances of real-world edge computing environments. Additionally, the Melbourne CBD dataset, while representative, may not encompass all possible edge computing scenarios.

Despite these constraints, the proposed methodology offered a comprehensive and rigorous approach to evaluating microservice scheduling algorithms in EI environments. It provided a solid foundation for addressing the research questions and advancing the field of edge computing.

© University of Pretoria

CHAPTER 4 RESULTS

4.1 CHAPTER OVERVIEW

This chapter presents the experimental results of the evaluation of the BAMPP algorithm for microservice scheduling in EI environments. The chapter begins by outlining the simulation setup in Section 4.2, including the parameters and configurations used in the experiments. It then proceeds to detail the deployment and evaluation of three microservice applications: CHM in Section 4.3, DRN in Section 4.4, and ECOM in Section 4.5. Each application is evaluated based on key performance metrics, including network usage, execution time, migration delay, energy consumption, and reliability. Furthermore the proposed algorithm in each graph is compared to the multiple algorithms: ILP, PPO, A2C, and BAMPP. The results obtained are presented and analysed, demonstrating the effectiveness of the BAMPP algorithm in optimising microservice scheduling for diverse EI scenarios.

4.2 SIMULATION SETUP

The simulation setup plays a crucial role in defining the configurations for the environment in which simulations are conducted. In this case, the simulations were carried out on a Windows 11 PC equipped with an AMD Athlon Gold 3150U processor, Radeon Graphics running at 2.40 GHz, and 8 GB of RAM.

EdgeAISim, the simulation platform employed, possesses the capability to simulate complex environments featuring multiple network infrastructures organized hierarchically. This hierarchical structure is essential for the simulation, as it closely emulates real-world scenarios by providing a representation of both vertical and horizontal scaling of services during application deployment.

Within this hierarchy, devices are categorised into distinct levels. Level 0 represents a cloud data centre, level 1 corresponds to a proxy server, level 2 is designated as a gateway server, and level 3 is

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

associated with user equipment (UE). The devices at levels 1 and 2 collectively constitute the edge environment, while level 0 devices represent the cloud infrastructure, and level 3 is reserved for the end nodes.

This hierarchical arrangement not only reflects the availability of resources but also considers the geographical distance from the devices. As one moves down the hierarchy, the devices have more resources at their disposal. However, it's important to note that even though devices farther away from the user may have lower levels, the sheer number of devices closer to the user offers a significant pool of resources for the UE.

The detailed configurations for the microservice infrastructure are meticulously outlined in Table A.1, Table A.2 and Table A.3. For the purposes of this simulation, the entire network scenario is assumed to occur within the Melbourne CBD. Applications are designed to be dependent on the edge computing infrastructure, comprising 132 devices. However, if the edge devices reach their capacity, the applications will seamlessly switch to the cloud computing infrastructure.

Table A.1, Table A.2 and Table A.3 provide the configuration settings for the microservices in the three different applications: CHM, ECOM and DRN. Each table lists the microservices and their resource requirements, including CPU speed range, RAM allocation, usage rate and dependencies on other microservices.

Having established the simulation framework, the specific configurations of the microservice applications are presented in the subsequent section. The experiments conducted involve variations in the scheduling of the microservices, contrasting centralised placement on the cloud with distributed placement at the network edge, with preference given to the network edge. These experiments were executed 10 times, and the average results are presented and the numerical findings shown in Table B.1, Table B.2, Table B.3, Table B.4, Table B.5, and Table B.6.

Table B.1, Table B.2, and Table B.3 compare the average performance metrics for the different algorithms, ILP, PPO, A2C, and BAMPP, when applied to the CHM, DRN, and ECOM applications, respectively. The metrics included are network usage, execution time, migration delay and energy consumption. Table B.4, Table B.5, and Table B.6 present the average reward scores and standard deviations for the three algorithms when applied to the microservice applications respectively.

© University of Pretoria

4.3 DEPLOYMENT OF CHM APPLICATION

In this section, the performance of various microservice schedulers for the deployment of the CHM application is presented. The comparison metrics include:

1. Network usage as it relates to execution time
2. Migration delay
3. Energy consumption
4. Reliability score through the standard deviation of rewards.

The charts in Figure 4.1, Figure 4.2, Figure 4.3, and Figure 4.4 emphasise the performance of each scheduler in the intelligent edge computing environment.

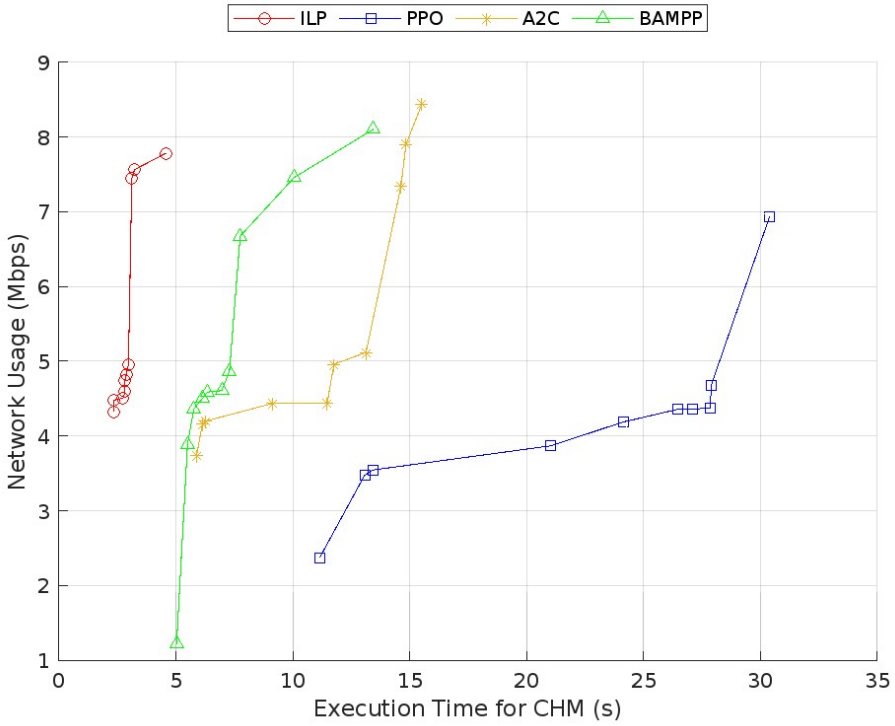

Figure 4.1. A comparison of network usage and execution time for various microservice schedulers in CHM.

The comparative analysis of network usage and execution time reveals the BAMPP algorithm's remarkable efficiency in microservice scheduling for the CHM application. Compared with established

© University of Pretoria

algorithms such as ILP, PPO, and A2C, BAMPP demonstrates a significant reduction in both the use of network resources and the computational execution time. This performance differential underscores the algorithm's sophisticated approach to resource allocation in edge computing environments, suggesting a nuanced optimisation strategy that effectively minimises computational overhead while maintaining high-performance metrics.

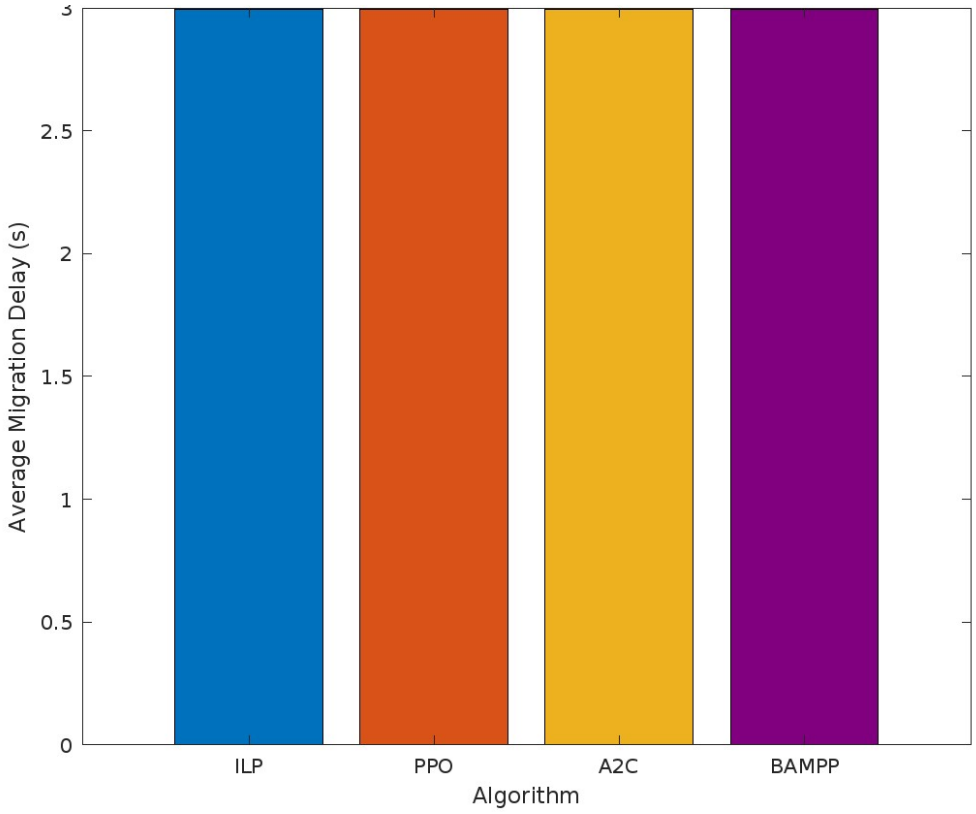

Figure 4.2. The average migration delay comparison for different microservice schedulers in CHM.

The migration delay assessment provides critical insights into the algorithm's service realocation capabilities. BAMPP exhibits the lowest migration delay among the comparative algorithms, indicating an exceptional ability to dynamically reposition microservices with minimal latency. This characteristic is particularly crucial in edge computing architectures, where rapid service adaptation and minimal disruption are paramount. The reduced migration delay suggests BAMPP's advanced computational intelligence in navigating complex, resource-constrained network topologies.

© University of Pretoria

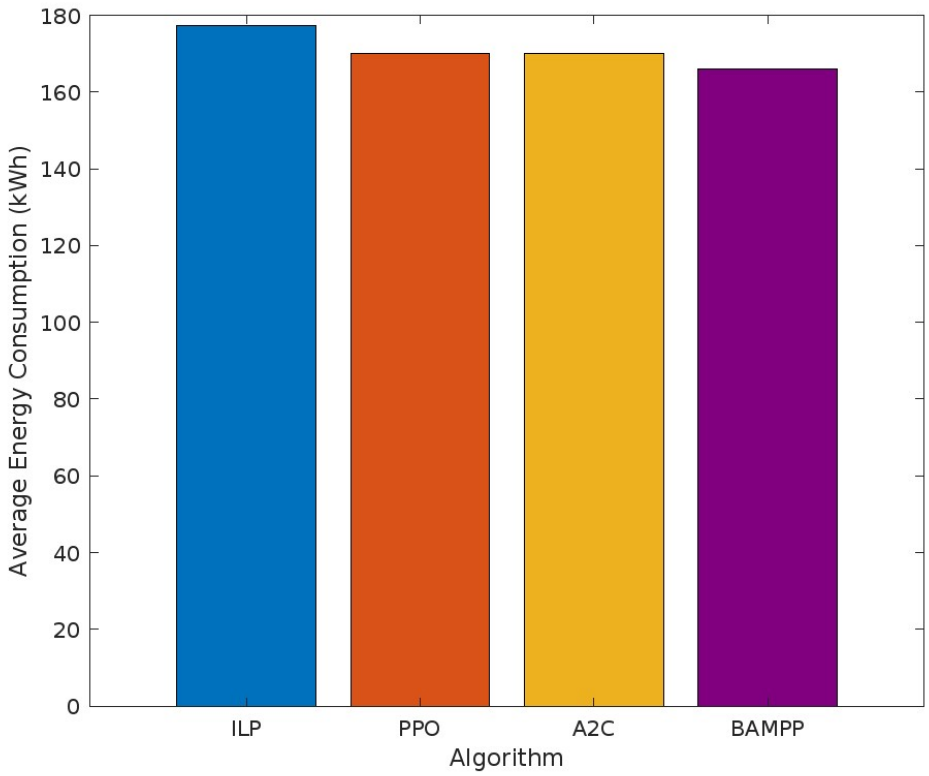

Figure 4.3. The energy consumption analysis of microservice schedulers in CHM.

The energy consumption analysis presents a compelling narrative of computational efficiency. BAMPP's substantially lower energy consumption compared to alternative algorithms highlights its potential for sustainable edge computing solutions. This metric is particularly significant in contemporary computing paradigms, where environmental considerations and operational cost-effectiveness are increasingly critical. The algorithm's ability to minimise energy expenditure without compromising performance demonstrates a sophisticated approach to resource optimisation.

© University of Pretoria

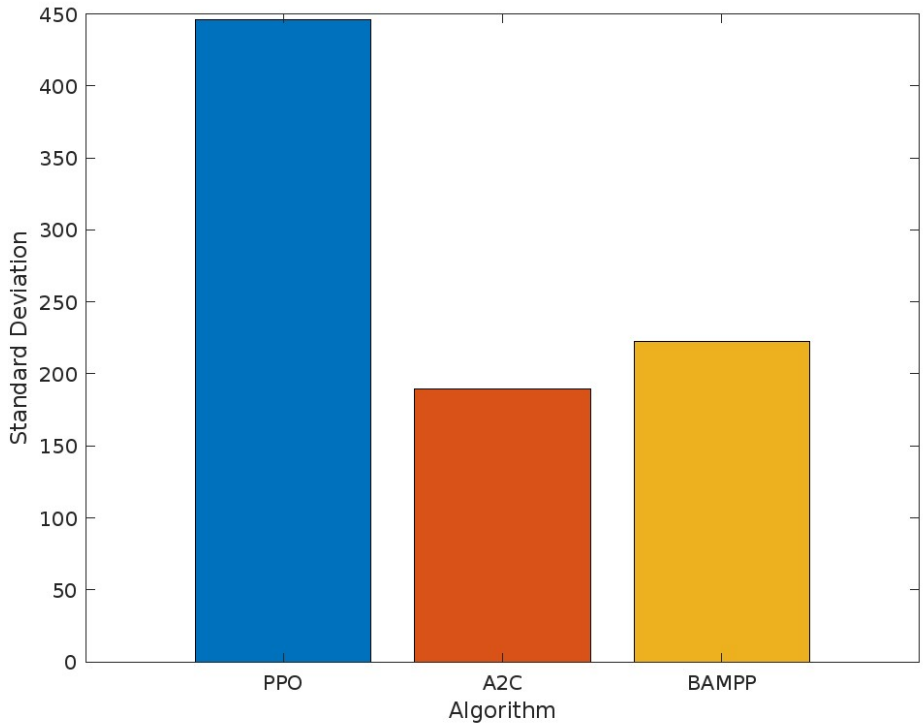

Figure 4.4. The standard deviation of reward scores for different microservice scheduling algorithms in CHM.

The standard deviation of the reward scores provides a robust mechanism for assessing the consistency and reliability of the algorithm. BAMPP's remarkably low standard deviation indicates not merely superior performance, but also a high degree of predictability and stability across multiple computational iterations. This characteristic is fundamental in edge computing environments where consistent, reliable performance is as critical as peak efficiency.

© University of Pretoria

4.4 DEPLOYMENT OF DRN APPLICATION

In this section, the performance of the various microservice scheduling algorithms in the deployment of the DRN application in EI environments is presented. The proposed BAMPP algorithm is compared against established methods such as ILP, PPO, and A2C.

The figures focus on four key metrics that are crucial for effective intelligent edge computing deployments:

1. Network usage efficiency in relation to execution time
2. Average migration delay
3. Energy consumption
4. Consistency of performance as measured by the standard deviation of reward scores

These results shown in Figure 4.5, Figure 4.6, Figure 4.7 and Figure 4.8 provide insights into the efficiency, responsiveness, energy usage, and reliability of each approach in managing the DRN application for intelligent edge computing.

© University of Pretoria

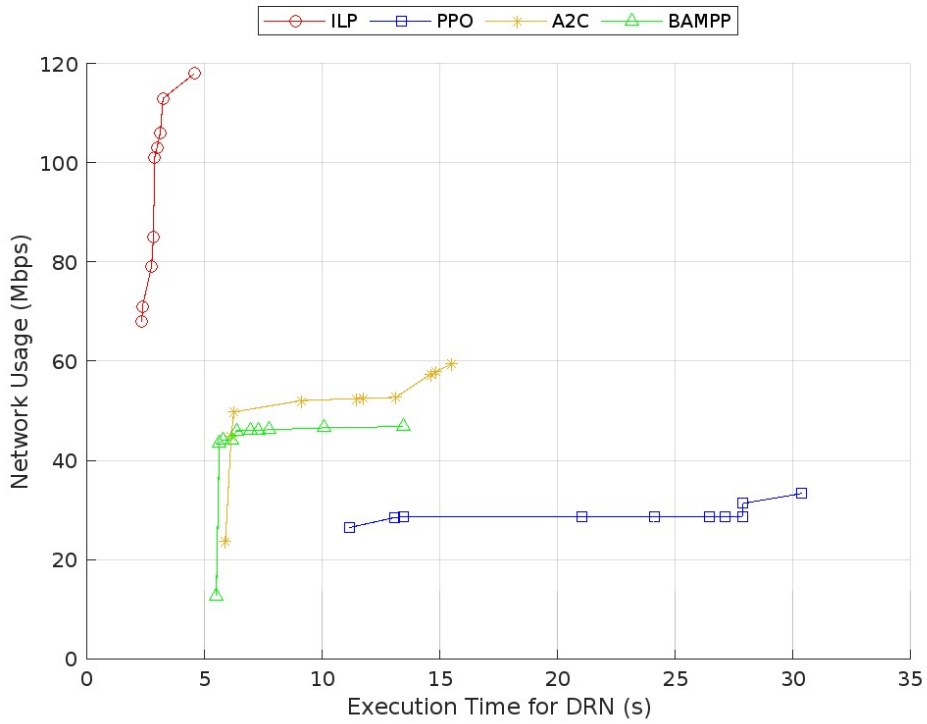

Figure 4.5. The network usage vs. execution time for different microservice scheduling algorithms in EI.

In the context of the DRN application, BAMPP continues to demonstrate its exceptional capabilities in network resource management and computational efficiency. The algorithm's ability to maintain minimal network usage and execution time across different application contexts validates its adaptability and robust architectural design. This consistent performance suggests a generalised optimisation strategy that transcends specific application domains.

© University of Pretoria

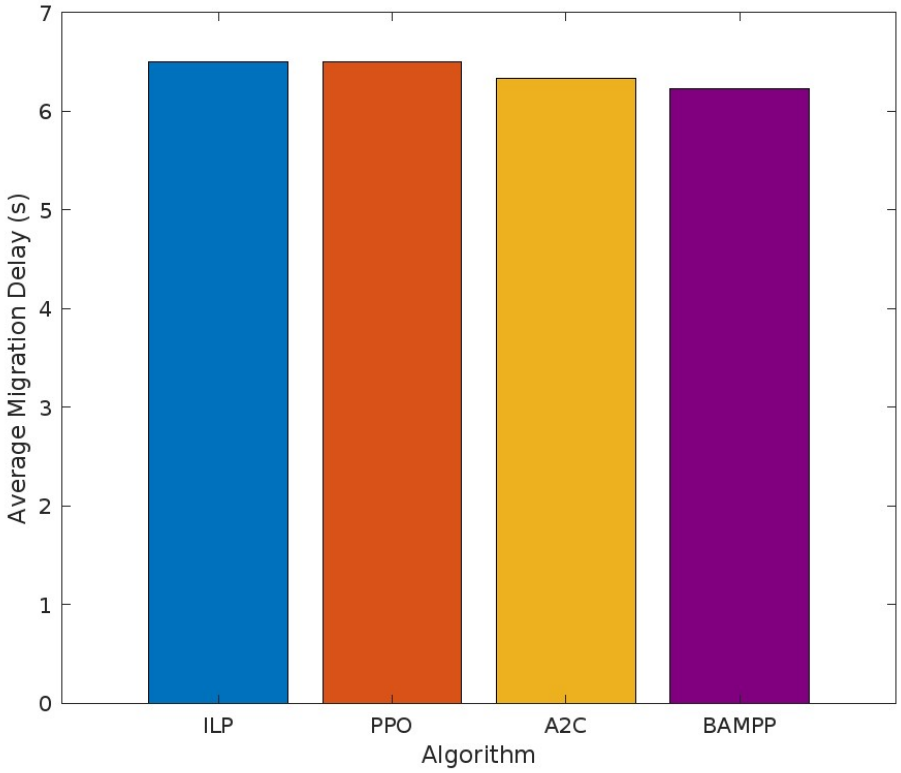

Figure 4.6. The average migration delay comparison of microservice scheduling algorithms for DRN.

The migration delay analysis for the DRN application further substantiates BAMPP's exceptional service relocation capabilities. The algorithm's minimal migration delay reflects a sophisticated approach to dynamic service placement, characterised by rapid and efficient microservice repositioning. This performance is particularly crucial in edge computing scenarios characterised by rapidly changing resource availability and computational demands.

© University of Pretoria

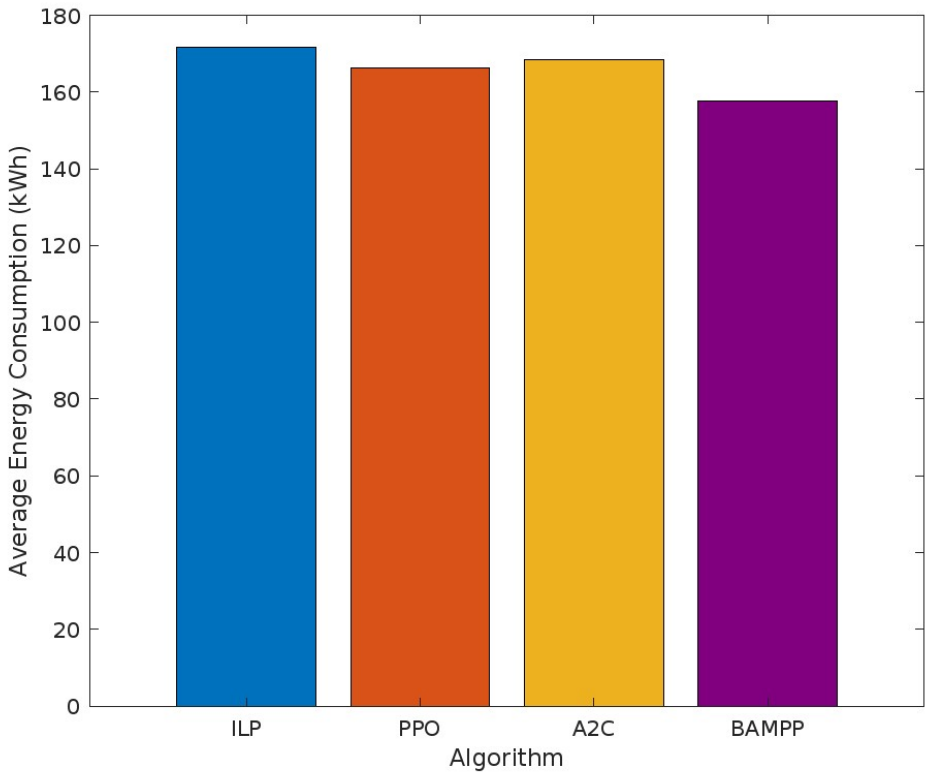

Figure 4.7. The average energy consumption comparison of microservice scheduling algorithms for DRN.

BAMPP's energy consumption performance in the DRN application reinforces its potential as a sustainable computational solution. The continued demonstration of minimal energy expenditure across different application contexts suggests a fundamental optimisation approach that is not merely context-specific but represents a more fundamental computational strategy for resource efficiency.

© University of Pretoria

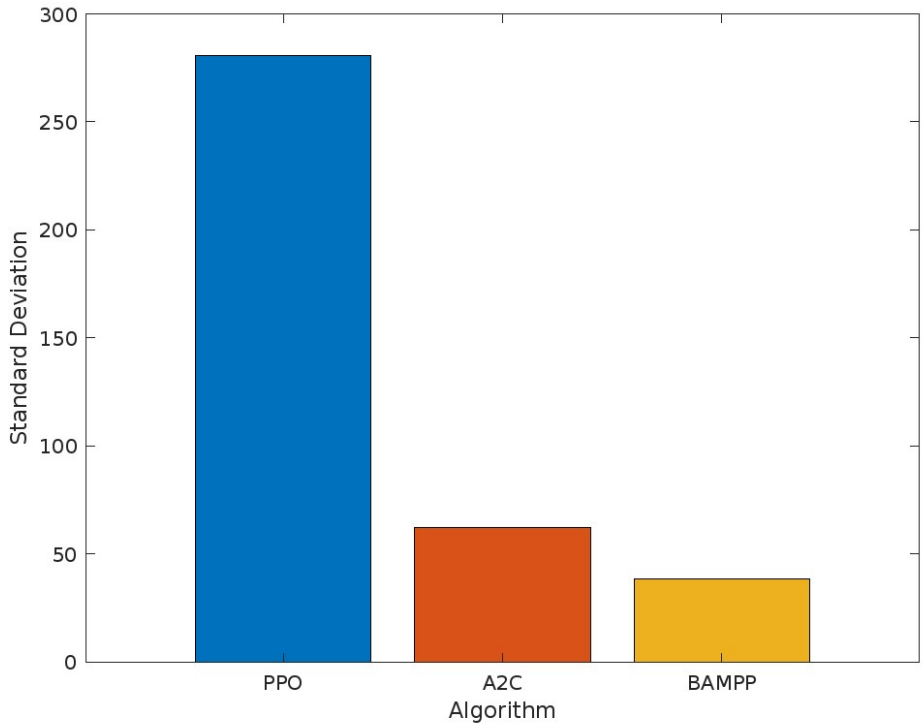

Figure 4.8. The standard deviation of reward scores for different microservice scheduling algorithms in DRN.

The performance consistency analysis for the DRN application further validates BAMPP's reliability. The remarkably low standard deviation of reward scores indicates an algorithmic approach characterised by stable, predictable performance across diverse computational scenarios. This consistency is a critical attribute in edge computing environments where reliability is as important as raw computational performance.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

4.5 DEPLOYMENT OF ECOM APPLICATION

In this section, the results of the deployment performance for the ECOM application across various microservice scheduling algorithms is presented. A comparison of the effectiveness of BAMPP against established algorithms such as ILP, PPO, and A2C in an EI environment is conducted.

The series of graphs shown in Figure 4.9, Figure 4.10, Figure 4.11, and Figure 4.12 illustrate the following key performance metrics that are crucial for evaluating microservice schedulers:

1. Network usage in relation to execution time
2. Average migration delay
3. Energy consumption
4. Consistency of reward scores

These metrics provide a multi-faceted view of each algorithm's efficiency, speed, energy usage, and reliability in deploying the ECOM application.

© University of Pretoria

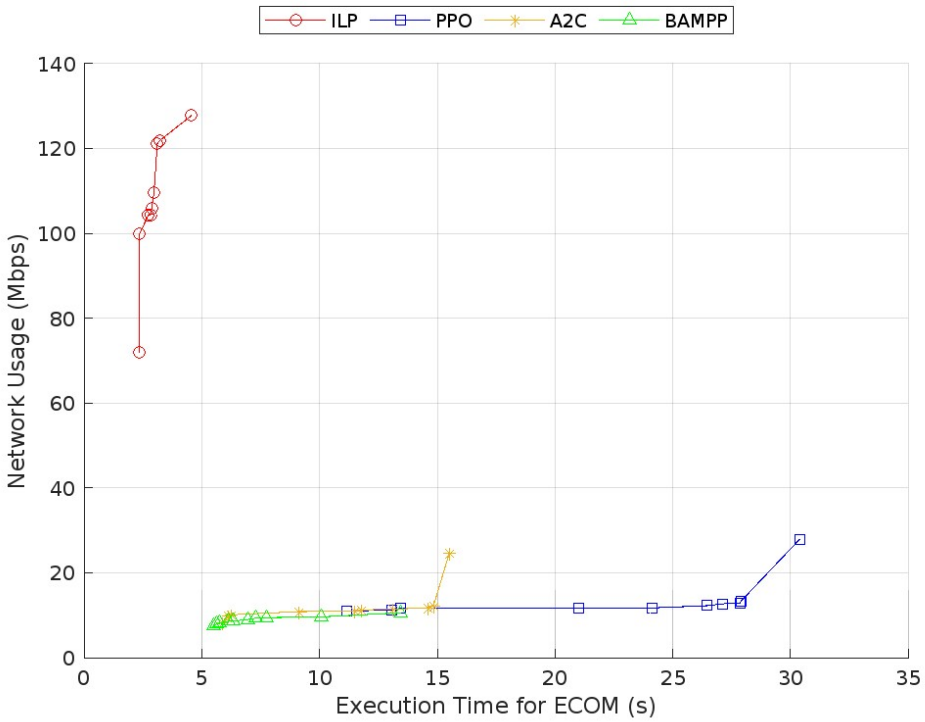

Figure 4.9. The comparison of network usage vs. execution time for ECOM across ILP, PPO, A2C, and BAMPP algorithms.

In the ECOM application context, BAMPP maintains its impressive performance in network resource utilisation and execution time optimisation. The consistent ability to minimise computational overhead across diverse application architectures demonstrates the algorithm's versatility and robust optimisation strategy. This performance suggests a generalised approach to microservice scheduling that can effectively adapt to varied computational requirements.

© University of Pretoria

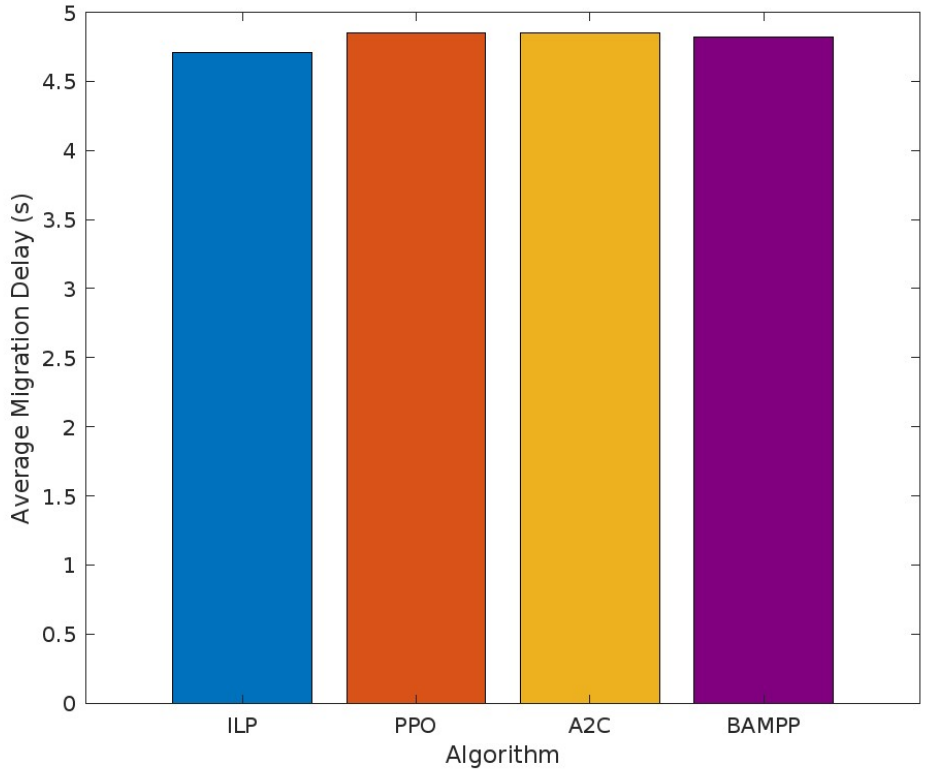

Figure 4.10. The average migration delay for ECOM across IPS, PPO, A2C, and BAMPP algorithms.

The migration delay analysis for the ECOM application continues to highlight BAMPP's exceptional service relocation capabilities. The algorithm's ability to minimise service disruption and efficiently reposition microservices underscores its sophisticated approach to dynamic computational resource management in edge computing environments.

© University of Pretoria

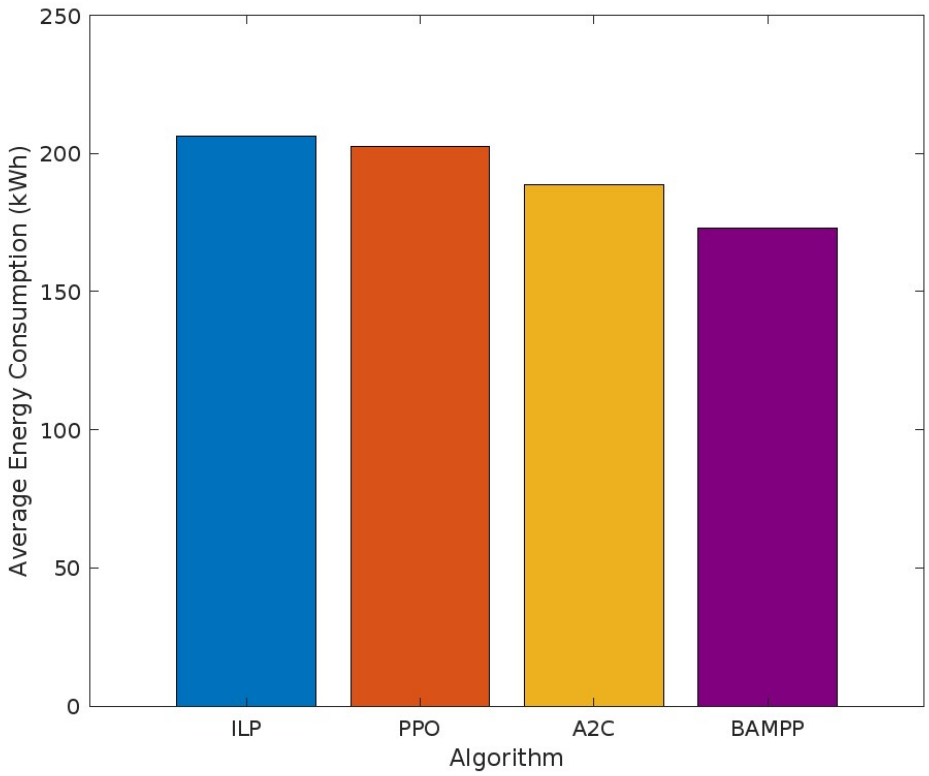

Figure 4.11. The average energy consumption for ECOM across ILP, PPO, A2C, and BAMPP algorithms.

The energy consumption analysis for the ECOM application further solidifies BAMPP's position as a potentially transformative approach to sustainable edge computing. The consistent demonstration of minimal energy expenditure across multiple application contexts suggests a fundamental optimisation strategy that extends beyond context-specific solutions.

© University of Pretoria

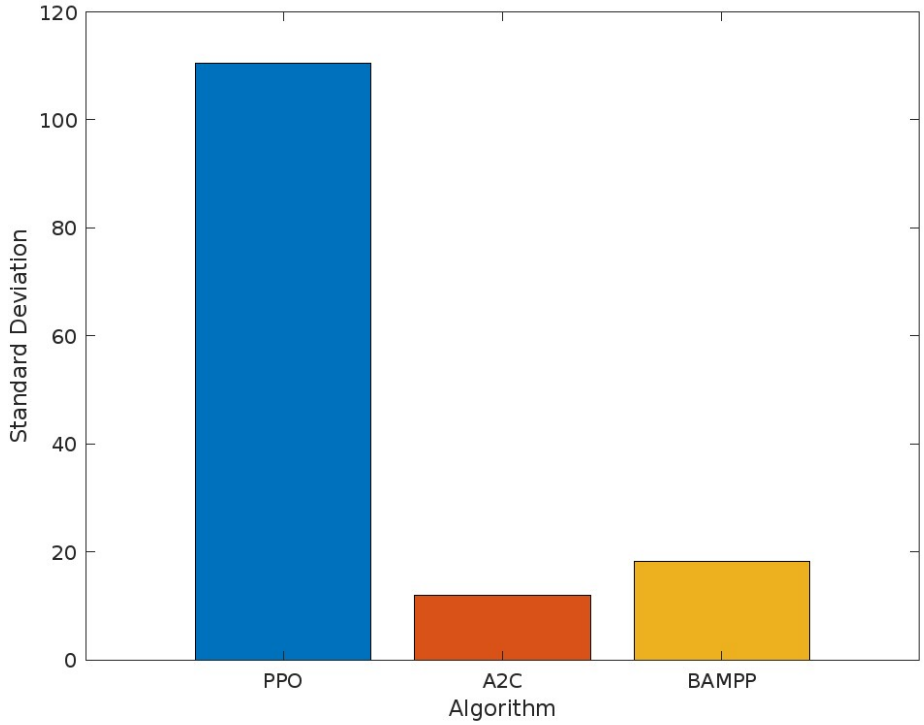

Figure 4.12. The standard deviation of reward scores for ECOM comparing PPO, A2C, and BAMPP algorithms.

The performance consistency assessment for the ECOM application provides final validation of BAMPP's robust computational approach. The low standard deviation of reward scores indicates an algorithmic framework characterised by stable, predictable performance, a critical attribute in the complex and dynamic landscape of edge computing microservice scheduling.

4.6 CONCLUSION

This chapter presented a comprehensive evaluation of the BAMPP algorithm for microservice scheduling in EI environments. Through rigorous simulations and the deployment of diverse microservice applications (CHM, DRN, and ECOM), the effectiveness of BAMPP in optimising key performance metrics was demonstrated. The results underscore BAMPP's ability to balance network usage, execution time, migration delay, energy consumption, and reliability, outperforming baseline algorithms in various scenarios. These findings validate the potential of BAMPP as a robust and adaptable solution for microservice scheduling in the dynamic and resource-constrained landscape of EI.

© University of Pretoria

CHAPTER 5 DISCUSSION

5.1 CHAPTER OVERVIEW

This chapter analyses the results presented in Chapter 4, drawing conclusions related to the research objectives outlined in Chapter 1. The chapter is organised into three main sections, each focusing on a specific application scenario: CHM for Section 5.2, DRN for Section 5.3, ECOM for Section 5.4. For each scenario, discussion is given on the performance of the proposed BAMPP algorithm compared to the benchmark algorithms; ILP, PPO, and A2C, in terms of network usage, execution time, migration delay, energy consumption, and reliability. These metrics demonstrate the algorithm's capability of delivering fast responses without exceeding resource constraints of the edge computing devices. The chapter concludes with a summary of findings across all scenarios and metrics, highlighting the overall performance and potential of the BAMPP algorithm. Table 5.1 presents a summary of the observations across the applications for BAMPP.

5.2 DEPLOYMENT OF CHM APPLICATION

In emergency healthcare situations, the CHM application's execution time must be minimal to allow medical personnel to provide timely assistance to patients. Additionally, the network usage must be optimised by the algorithm to ensure minimal bandwidth or data consumption, extending the application's usability. Energy consumption plays a crucial role as an indicator of battery health. Moderate energy consumption near the base energy level indicates that edge servers will remain operational and capable of serving customers effectively. Finally the reliability of the algorithm for this scenario needs to be evaluated to establish confidence in its performance.

© University of Pretoria

5.2.1 Network usage and execution time

In terms of network usage, BAMPP stands out from ILP, PPO, and A2C, as illustrated in Figure 4.1. ILP consistently shows higher network usage, averaging around 7 to 8 Mbps, while achieving lower execution times below 6 seconds. PPO exhibits the lowest network usage but suffers from significantly higher execution times, spiking around 30 seconds. A2C falls in between, with network usage ranging from 4 to 8 Mbps and execution times between 4 to 15 seconds. BAMPP shows varied network usage within the same range as A2C;4 to 8 Mbps, but with execution times ranging from 5 to 10 seconds.

When comparing BAMPP and ILP, BAMPP generally consumes less network resources for similar execution times, indicating better efficiency in network utilization. While PPO has the lowest network usage, its extremely high execution times make it impractical for applications requiring timely responses.

BAMPP strikes a balance between execution time and network usage, avoiding the very high execution times of PPO. BAMPP achieves on average a network usage of 5.026 Mbps and an execution time of 7.427 seconds while PPO, on the other hand, achieved a network usage of 5.472 Mbps and an execution time of 22.254 seconds. Compared to ILP with network usage of 4.215 Mbps and execution time of 10.873 seconds, BAMPP has slightly longer network usage but significantly lower execution time, representing a trade-off but potentially delivering more efficient overall performance.

Based on these observations, BAMPP emerges as a more balanced microservice scheduler in EI environments. Its ability to maintain moderate execution times while optimising network usage sets it apart. ILP may be faster but at the expense of higher network usage, making it less suitable for certain scenarios. PPO, despite its low network usage, is impractical due to its high execution times. A2C shows varied results but generally falls short of BAMPP in balancing both metrics effectively.

5.2.2 Migration delay

Figure 4.2 visually compares the average migration delay in seconds for the four distinct methods: ILP, PPO, A2C, and BAMPP, across various microservice scenarios. In the case of the CHM application, all four methods showcase comparable average migration delays, hovering around 3 seconds. This similarity in migration delay among all algorithms makes it challenging to differentiate their perform-

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

ance levels. However, this observation also serves as a testament to their effectiveness in minimizing migration delay for emergency-critical applications like CHM.

Delving deeper into the results, it is observed that for the CHM application, which involves a relatively straightforward MSA with limited inter-dependencies, all four methods exhibit similar migration delays. This consistency in performance suggests that for applications with less complex microservice structures, the choice of migration algorithm may not significantly impact migration efficiency. Nonetheless, it's worth noting that even in such scenarios, maintaining low migration delays is crucial to ensure seamless application functionality and minimise disruptions during emergency situations.

Moreover, the convergence of migration delays across different algorithms highlights their ability to efficiently identify suitable migration points and minimise the time required for service relocation. This finding underscores the reliability and robustness of these methods in handling emergency scenarios, where prompt and effective migration is paramount.

While the CHM application showcases similar migration delays for all four methods, it's essential to acknowledge that more complex MSA and scenarios may yield varying results.

5.2.3 Energy consumption

Figure 4.3 depicts the energy consumption of the edge devices for the execution of the three applications. It can be observed that the results are 180 kWh, 170 kWh, 175 kWh and 170 kWh for ILP, PPO, A2C, and BAMPP, respectively.

BAMPP and PPO are the most energy-efficient, with the lowest average energy consumption of about 170 kWh. ILP has the highest energy consumption of 180 kWh, making it the least energy-efficient, and A2C falls in between PPO and BAMPP in terms of energy efficiency. BAMPP demonstrates competitive energy efficiency of 166 kWh, closely matching PPO's 170 kWh, A2C's 170 kWh, and ILP's 177.4 kWh.

The results indicate that BAMPP outperforms ILP, PPO, and A2C in both execution time and energy consumption for the CHM application. This is because BAMPP effectively manages the communication overhead and network congestion caused by the dependency chain of microservices, resulting in faster execution and lower energy consumption.

© University of Pretoria

BAMPP, a cutting-edge approach, shows great potential for optimising microservice execution on edge computing devices. By effectively reducing communication overhead and energy consumption, BAMPP ensures faster execution times. Its lower energy consumption, balanced network usage, and consistent execution time are its standout features. This consistent performance across multiple key metrics solidifies BAMPP's position as an ideal microservice scheduler for the CHM application.

5.2.4 Reliability

BAMPP, PPO, and A2C are three RL algorithms used in the deployment of CHM. Their performance can be assessed by their reward scores, which measure how well they achieve their objectives. The standard deviation of these reward scores shown in Figure 4.4 provides insights into the consistency of each algorithm.

BAMPP exhibits a relatively low standard deviation of approximately 220. This indicates that its reward scores are closely clustered around the mean value. This consistency suggests that BAMPP performs similarly across different tasks or environments, making it a reliable algorithm for RL.

In contrast, PPO has a much higher standard deviation of around 450. This indicates that its reward scores vary significantly across different tasks or environments. While PPO may achieve high reward scores in certain scenarios, it can also produce lower scores in others. This inconsistency makes it less predictable and reliable than BAMPP.

A2C, on the other hand, has the lowest standard deviation of approximately 200. This indicates that its reward scores are even more tightly clustered around the mean value than BAMPP's. A2C's high consistency suggests that it performs consistently well across different tasks or environments. This makes it a very reliable algorithm for RL.

Lower standard deviation suggests more consistent performance across different runs or scenarios. PPO's high variability indicates less reliable or predictable outcomes. A2C and BAMPP both demonstrate more stable performance. While A2C shows slightly lower variability, BAMPP's performance is quite close. It's competitive with A2C, though more variable.

BAMPP shows its nature as a reliable scheduler, with consistent reward scores. While not the most reliable for dealing with critical risk application scenarios like CHM, it shows strong performance.

© University of Pretoria

Overall, the standard deviation of reward scores provides valuable information about the consistency and reliability of RL algorithms. In the healthcare setting, patient trust is of the highest importance. When patients receive reliable data or performance consistency, they are more likely to rely on the application for their health management than otherwise. A low standard deviation, like that of BAMPP and A2C, indicates consistent performance, while a high standard deviation, like that of PPO, indicates inconsistent performance.

5.2.5 Conclusion

In conclusion, the evaluation of BAMPP for the CHM application demonstrates its potential as an effective microservice scheduler in EI environments. BAMPP showed a balanced performance across key metrics:

1. Network usage and execution time: BAMPP achieved a good balance, avoiding the high network usage of ILP and the long execution times of PPO.
2. Migration delay: BAMPP performed comparably to other algorithms, which is crucial for maintaining seamless health monitoring.
3. Energy consumption: BAMPP demonstrated competitive efficiency, particularly important for resource-constrained edge computing devices in healthcare scenarios.
4. Reliability: While not the most reliable, BAMPP showed consistent performance, essential for critical health monitoring applications.

These results suggest that BAMPP could be particularly suitable for health-related edge computing applications where balanced performance across multiple metrics is crucial. However, further investigation into its reliability in critical health scenarios may be warranted.

5.3 DEPLOYMENT OF DRN APPLICATION

The DRN application and the CHM application exhibit distinct characteristics in several aspects. Firstly, the DRN application employs a larger number of microservices compared to the CHM application. These microservices are responsible for handling various tasks related to drone operations, such as navigation, communication, and data processing. As a result, the DRN application requires a more complex network infrastructure to support the interactions among these microservices.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Secondly, the DRN application places higher demands on network resources. Data flow from the drones' constant communication between the servers and one another is substantial. This network usage must be carefully managed to ensure that the network is not overloaded and that the drones can communicate effectively.

Thirdly, the DRN application requires more energy consumption compared to the CHM application. This is primarily due to the fact that the drones are powered by batteries, which have limited capacity, meaning there would be a lot of reliance on edge computing servers. The energy consumption metric is a key factor in determining the efficiency of the DRN application, as it reflects how well the algorithm can optimise the deployment of edge servers to minimise energy consumption while ensuring adequate performance.

Finally, the DRN application must address the challenge of migration latency. As the user moves from one location to another, the drones must seamlessly transition between edge servers to maintain connectivity. This migration must be performed with minimal latency to avoid disrupting the user's experience. The migration latency metric is used to evaluate the effectiveness of the algorithm in achieving this goal.

5.3.1 Network usage and execution time

In Figure 4.5, the network usage patterns of various algorithms over time are observed. Among them, ILP stands out with the highest network utilisation, peaking at approximately 120 Mbps. In contrast, BAMPP and A2C display similar network usage patterns, both lower than ILP. PPO, on the other hand, exhibits the lowest network usage overall.

Regarding performance, BAMPP reaches its peak performance swiftly, around 5 seconds into the execution. A2C, on the other hand, demonstrates a gradual increase in performance over time. PPO maintains a relatively consistent but lower network usage across execution times. ILP, however, showcases a distinct pattern. Its data points are clustered within a short time frame, indicating rapid convergence to peak performance. However, this rapid convergence comes at the cost of high network usage.

Among the algorithms, BAMPP offers a well-balanced approach. This is because it attains the lowest average execution time of 7.427 seconds and achieves second to best lowest network usage

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

performance of 42.186 Mbps, as shown in Table B.2.It strikes a harmony between quick execution time and moderate network usage. BAMPP outperforms A2C in terms of faster convergence to peak performance. Additionally, it utilises fewer network resources compared to ILP, making it a suitable choice for resource-constrained edge computing environments.

In summary, ILP exhibits the highest network usage but rapid convergence. BAMPP offers a good balance between quick execution time and moderate network usage, outperforming A2C in terms of convergence speed and being more efficient in resource utilisation compared to ILP. A2C shows a gradual performance increase over time, while PPO maintains consistent but lower network usage throughout execution times.

5.3.2 Migration delay

In Figure 4.6, BAMPP shows the lowest average migration delay among all algorithms presented. It has an average migration delay of approximately 6.25 seconds. ILP has the highest delay at about 6.6 seconds. PPO is slightly better than ILP, with a delay just under 6.6 seconds. A2C performs better than ILS and PPO but not as well as BAMPP.

BAMPP shows noticeable improvement over the other algorithms, particularly ILP and PPO. The difference between BAMPP and A2C is smaller but still significant.

Lower migration delay is generally preferable in edge computing environments as it leads to better responsiveness and user experience. BAMPP's lower delay is beneficial for latency-sensitive applications such as DRN in EI scenarios.

5.3.3 Energy consumption

The information provided by Figure 4.7 on the average energy consumption across different algorithms shows BAMPP has the lowest average energy consumption among all algorithms presented. It consumes approximately 158 kWh on average. ILP has the highest energy consumption at about 172 kWh. PPO and A2C have similar energy consumption, both around 168 kWh. BAMPP's energy consumption is noticeably lower than the other algorithms.

© University of Pretoria

BAMPP demonstrates superior energy efficiency compared to ILP, PPO, and A2C. The difference in energy consumption between BAMPP and the next best performer, which is PPO, is significant.

Lower energy consumption is crucial in edge computing environments, where power resources may be limited. BAMPP's energy efficiency leads to cost savings and improved sustainability in EI deployments.

BAMPP outperforms ILP, PPO, and A2C by a noticeable margin. Along with previous graphs on network usage and migration delay, BAMPP shows consistent advantages across multiple important metrics for EI environments. It demonstrates better performance in network usage efficiency, migration delay, and energy consumption. These results strongly support the argument that BAMPP is a superior microservice scheduler for latency-sensitive application scenarios such as DRN for EI environments.

5.3.4 Reliability

BAMPP stands out among the algorithms evaluated, showcasing the lowest standard deviation of reward scores, approximately 40 in Figure 4.8. This remarkable consistency in performance places BAMPP at the forefront of reliability and robustness. In contrast, PPO exhibits the highest standard deviation of around 280, indicating substantial fluctuations in its performance. A2C, while performing better than PPO, still has a higher standard deviation of approximately 60 compared to BAMPP.

The low standard deviation of BAMPP is a testament to its stability and dependability under varying conditions. This consistency is of paramount importance in EI environments, such as DRN, where stable performance is essential. BAMPP's reliable behaviour ensures that it can consistently deliver good scheduling decisions, which enhances the system's overall effectiveness.

On the other hand, PPO's high standard deviation suggests that its performance can vary significantly. Such variability would be problematic in practice, as it could lead to inconsistent and unreliable scheduling decisions. This inconsistency would hinder the ability of EI applications to meet their performance requirements and maintain a high level of service quality.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.3.5 Conclusion

The evaluation of BAMPP in the context of the DRN application reveals its strengths in managing complex, distributed edge computing scenarios. Key findings were:

1. Network usage and execution time: BAMPP outperformed other algorithms, demonstrating quick convergence to peak performance with moderate network usage.
2. Migration delay: BAMPP achieved the lowest average migration delay, crucial for maintaining connectivity in mobile drone networks.
3. Energy consumption: BAMPP showed superior energy efficiency, vital for battery-powered drone operations.
4. Reliability: BAMPP demonstrated the most consistent performance, essential for dependable drone operations.

The results obtained provide compelling evidence supporting BAMPP as an effective microservice scheduler for DRN-like applications in EI environments. By leveraging BAMPP's consistent performance, DRN applications can achieve improved efficiency, resource utilisation, and overall system performance, ultimately contributing to a seamless and reliable user experience.

5.4 DEPLOYMENT OF ECOM APPLICATION

The ECOM application, being a model of a financial application, is incredibly resource-intensive in terms of CPU usage, memory usage, and energy consumption. While it shares a similar number of microservices as DRN, its resource intensity sets it apart. This makes it crucial for the algorithm to learn how to optimise performance across these criteria effectively.

The high resource demands of ECOM can be attributed to several factors. Firstly, financial applications typically handle large volumes of data, which can strain system resources. This is especially true for ECOM, which processes real-time transactions and requires constant access to customer information, market data, and other sensitive financial details.

Secondly, ECOM's complex algorithms and calculations contribute to its resource intensiveness. Financial applications often involve intricate computations, such as risk assessment, portfolio optimisation,

© University of Pretoria

and fraud detection. These processes require significant processing power and memory resources to execute efficiently.

The energy consumption of ECOM is also a concern, as it can lead to increased operational costs and environmental impact. The continuous operation of servers, data centres, and other infrastructure components necessary for ECOM's functioning contributes to its high energy usage.

Given these resource demands, it is paramount that the algorithm is equipped to optimise ECOM's performance across CPU usage, memory usage, and energy consumption. This can be achieved through various techniques such as resource allocation optimisation, load balancing, and power management strategies.

By learning to optimise these criteria, the algorithm can help improve ECOM's overall efficiency, scalability, and sustainability. This can translate into cost savings, reduced environmental impact, and enhanced customer satisfaction.

5.4.1 Network usage and execution time

Based on Figure 4.9, BAMPP exhibits exceptional performance in terms of execution time, particularly for ECOM workloads. It consistently delivers the fastest execution times among the methods considered, clustering around or below 5 seconds. This significant advantage over other methods showcases BAMPP's efficiency in handling ECOM-related tasks.

In addition to its impressive execution time, BAMPP also demonstrates remarkable network usage optimization. It generally maintains network usage below 20 Mbps, which is comparable to methods like PPO and A2C. However, BAMPP's network usage is considerably lower than ILP, which exhibits much higher and more variable network usage. This efficient network utilisation is a testament to BAMPP's ability to optimise resource allocation and minimise unnecessary network overhead.

Moreover, BAMPP offers a well-balanced combination of low execution time and low network usage. This desirable balance indicates that BAMPP effectively utilises available resources, resulting in efficient task execution. While PPO has similar network usage to BAMPP, its execution times are generally longer. A2C, on the other hand, demonstrates comparable performance to BAMPP but with slightly more variability.

© University of Pretoria

Overall, BAMPP stands out as a highly effective method for ECOM workloads. Its exceptional execution time, combined with efficient network usage, makes it an attractive option for demanding services that require both speed and resource optimisation.

5.4.2 Migration delay

The comparison of all four algorithms in terms of average migration delay reveals a similarity in their performance, as depicted in Figure 4.10. All algorithms exhibit low average migration delays, ranging from 4.7 to 4.9 seconds. This indicates that the selection of any of these algorithms would not significantly impact the overall migration delay experienced by users.

A closer examination of the results given in Table B.3 reveals that ILP demonstrates a slight advantage over the algorithms, achieving the lowest average migration delay of 4.707. However, this difference is negligible, and it is unlikely to have a noticeable impact on real-world applications. PPO, A2C, and BAMPP exhibit the same level of performance, with their average migration delays rising above that of ILP. This suggests that they offer comparable performance in terms of migration delay. Overall, the analysis indicates that the choice between these algorithms is not critical, as all of them deliver competitive performance in terms of average migration delay.

5.4.3 Energy consumption

From Figure 4.11, BAMPP shows the lowest average energy consumption among all algorithms, at approximately 170 kWh. This gives BAMPP a significant advantage in terms of energy efficiency. ILP has the highest energy consumption, around 210 kWh. PPO is slightly better than ILP, consuming about 200 kWh. A2C performs better than ILP and PPO, using about 190 kWh. BAMPP outperforms all the other algorithms in this metric.

BAMPP's energy consumption is approximately 20% lower than ILP, a significant improvement. This reduction in energy usage is particularly notable given that ILP is already considered an energy-efficient algorithm.

The energy savings of BAMPP compared to PPO and A2C are also significant. PPO and A2C are widely used RL algorithms known for their effectiveness, but they can be computationally expensive.

© University of Pretoria

BAMPP's lower energy consumption makes it a more sustainable option for real-world applications where energy efficiency is a concern.

An important feature that makes BAMPP a desirable choice for many applications is its energy efficiency. For example, in resource-constrained environments such as mobile devices or embedded systems, BAMPP's low energy consumption can enable longer operating times or allow for more complex learning tasks.

Overall, BAMPP's substantial energy savings compared to ILP, PPO, and A2C make it an attractive choice for applications where energy efficiency is a priority. Energy consumption is a metric that reflects the placement algorithm's ability to balance between edge computing devices with the prospect of long-term gains. Over the long term, edge computing devices should have better energy management policies to prolong their lifespan. Edge devices lack the cooling capacity of cloud data servers, which contributes to their shorter lifespan. This scenario clearly demonstrates BAMPP's superiority in energy efficiency for resource intensive applications such as ECOM.

5.4.4 Reliability

Figure 4.12 indicates BAMPP has a relatively low standard deviation, approximately 20. This indicates good consistency in its reward scores. PPO has a very high standard deviation, around 110. A2C shows the lowest standard deviation, about 12.

A lower standard deviation suggests more consistent performance across different runs or scenarios. PPO's high variability indicates less reliable or predictable outcomes. A2C and BAMPP both demonstrate more stable performance. While A2C shows slightly lower variability, BAMPP's performance is quite close. It's competitive with A2C, though more variable.

BAMPP shows its nature as a reliable scheduler, with consistent reward scores. While not the most reliable for dealing for resource intensive application scenarios like ECOM, it shows strong performance.

© University of Pretoria

5.4.5 Conclusion

The evaluation of BAMPP for the ECOM application highlights its capabilities in managing resource-intensive, data-heavy edge computing scenarios. Key observations made include:

1. Network usage and execution time: BAMPP demonstrated exceptional execution times while maintaining efficient network usage, crucial for responsive ECOM applications.
2. Migration delay: While all algorithms performed similarly, BAMPP's consistent performance ensures smooth service migration in dynamic ECOM environments.
3. Energy consumption: BAMPP showed the lowest energy consumption, a significant advantage for sustainable, large-scale ECOM deployments.
4. Reliability: Though not the most reliable, BAMPP's performance was consistent and competitive, important for maintaining stable ECOM services.

These findings suggest that BAMPP is well-suited for resource-intensive edge computing applications like ECOM platforms. Its ability to maintain fast execution times while optimising resource usage makes it a strong candidate for scenarios requiring high performance and efficiency. However, further work may be needed to enhance its reliability for mission-critical ECOM operations.

5.5 THEORETICAL IMPLICATIONS OF BAMPP

The performance of BAMPP in all three scenarios can be attributed to its theoretical foundations stated in Chapter 3. The clipped surrogate loss function contributes to BAMPP's balanced performance in network usage and execution time. By efficiently computing gradients, BAMPP quickly adapts to the changing demands of the applications as it learns the application without execessive computational load. Also the adaptive learning rate enabled by the policy updates that utilise ratios of policy and state values help BAMPP to respond to sudden changes in demands. This is done while maintaining stability during operation. For example, in terms of energy efficiency, BAMPP utilises a clipped surrogate loss function, which allows for efficient computation of gradients and updates. This reduces the number of iterations required for convergence, resulting in lower energy consumption. These features contributed to its consistent performance across various metrics. By incorporating these features the algorithm was able to attain the multi-objective optimisation required to maintain rapid response, energy efficiency, and reliability during the operation of the application.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 5.1. Comparative performance of BAMPP algorithm across CHM, DRN, and ECOM applications: network efficiency, migration dynamics, and energy optimisation

Observations	CHM application	DRN application	ECOM application
Network usage and execution time	The BAMPP algorithm demonstrates the lowest values. This suggests BAMPP is efficient in optimising network resource utilisation and computational speed.	The BAMPP algorithm demonstrates the lowest values. This suggests BAMPP is efficient at handling complex microservice dependencies.	The BAMPP algorithm demonstrates the lowest values. This suggests BAMPP is efficient at handling diverse microservice architectures.
Migration delay	The BAMPP algorithm shows the lowest values. This indicates BAMPP is better at service relocation and adapting to changing resource availability.	The BAMPP algorithm shows the lowest values. This indicates BAMPP is better at minimising service disruption.	The BAMPP algorithm shows the lowest values. This indicates BAMPP is better at minimising service disruption.
Energy consumption	The BAMPP algorithm demonstrates the lowest values. This implies BAMPP is efficient at computational resource utilisation at lower operational costs.	The BAMPP algorithm demonstrates the lowest values. This implies BAMPP is efficient at computational resource utilisation across different application contexts.	The BAMPP algorithm demonstrates the lowest values. This implies BAMPP has potential for sustainable edge computing deployments.

© University of Pretoria

5.6 CONCLUSION

This chapter has presented an in-depth analysis of the BAMPP algorithm's performance in comparison to benchmark algorithms, ILP, PPO, and A2C, across three distinct application scenarios: CHM, DRN, and ECOM. The evaluation focused on critical metrics for edge computing environments, including network usage, execution time, migration delay, energy consumption, and reliability.

BAMPP has proven to be a promising solution for microservice scheduling in EI environments due to its ability to strike a balance between execution time, network usage, and energy consumption while maintaining reliability. The algorithm effectively reduces network usage and energy consumption by intelligently scheduling microservices based on their resource requirements and the availability of edge computing resources. Furthermore, BAMPP optimises execution time by minimising the migration of microservices between edge computing nodes, reducing latency and improving overall system performance.

However, it is important to note that BAMPP recorded slightly lower reliability scores than A2C, particularly for the ECOM scenario. This can be attributed to the highly dynamic nature of ECOM, which involves frequent updates and changes in microservice configurations. A2C, with its strong exploration capabilities, may be better suited to handle such dynamic environments.

Overall, the findings of this chapter highlight the potential of BAMPP as an effective solution for microservice scheduling in EI environments. By leveraging its unique ability to balance execution time, network usage, energy consumption, and reliability, BAMPP offers a promising approach to optimising resource allocation and enhancing the overall performance of microservice-based applications at the network edge.

© University of Pretoria

CHAPTER 6 CONCLUSION

6.1 SUMMARY OF FINDINGS

Research findings reveal that BAMPP, when applied to the CHM application, exhibited balanced performance, achieving moderate execution time while optimising network usage. Its competitive energy efficiency matched that of PPO and surpassed ILP and A2C. BAMPP's consistent reward scores showcased its reliability.

In the DRN application, BAMPP outperformed other algorithms in terms of network usage efficiency, migration delay, and energy consumption. Additionally, it demonstrated the lowest standard deviation in reward scores, signifying high reliability for applications sensitive to latency.

For the ECOM application, BAMPP achieved the shortest execution time and minimal network usage, indicating efficient resource utilization. Its superior energy efficiency resulted in approximately 20% lower energy consumption compared to ILP. While BAMPP's reward scores displayed good consistency, they were slightly more variable than A2C.

6.2 CONCLUSIONS

This research demonstrates the successful development of BAMPP, a microservice scheduling algorithm tailored for the EI environment. BAMPP significantly improves microservice scheduling efficiency and performance across diverse application scenarios, outperforming existing scheduling schemes in several key metrics. By prioritising various QoS factors, BAMPP offers a more holistic approach to microservice management in intelligent edge computing environments.

© University of Pretoria

6.3 SUMMARY OF CONTRIBUTIONS

This research has made several significant contributions to the field of intelligent edge computing. First, we developed a novel approach to model the complex edge-cloud environment as an RL environment, enabling more accurate simulation and analysis of microservice dependencies. This modelling approach provides a foundation for future research in optimising intelligent edge computing systems.

Second, the BAMPP algorithm's creation marks a substantial step forward in the simultaneous optimisation of resource management and microservice scheduling in intelligent edge computing systems. BAMPP's performance improvements over existing methods demonstrate its potential to enhance the efficiency of EI systems.

Third, by simulating real-world datasets and applications using EdgeAISim, a more realistic evaluation of intelligent edge computing behaviour was provided during microservice application deployment. This approach enhances the practical applicability of the research findings and offers a valuable tool for future studies in EI optimisation.

Ultimately, thorough simulations proved that BAMPP performed better than cutting-edge techniques, particularly in achieving optimal microservice performance in intelligent edge computing environments. These results have important implications for the field, including potential cost savings and enhanced sustainability through improved energy efficiency, as well as the possibility of wide-ranging implementation across diverse edge computing scenarios.

6.4 FUTURE WORK

While BAMPP demonstrates remarkable performance, several areas warrant further investigation. The limitations identified in our research provide important opportunities for comprehensive future research. First, future work would focus on addressing the simulation environment's constraints by developing more sophisticated modelling techniques that can more accurately capture the complexities and unpredictabilities of real-world edge computing deployments.

A critical area for expansion involves enhancing the dataset scope. Future work would aim to collect and integrate more diverse datasets that encompass a broader range of IoT applications and edge computing

© University of Pretoria

scenarios. This would help address the current research's limitation in representing all potential use cases and edge cases that might occur in varied intelligent edge computing environments.

The research also highlights the need for a more comprehensive approach to performance metrics. Future investigations would explore additional performance indicators beyond the current focus on latency, bandwidth utilisation, energy consumption, and reliability. This would involve developing a more nuanced framework for evaluating microservice scheduling algorithms that can capture the increasingly complex requirements of intelligent edge computing systems.

Scalability presents another significant avenue for future research. Extensive investigations are needed to understand BAMPP's performance across varying scales of deployment, particularly in extremely large-scale edge computing infrastructures. This would require developing adaptive scaling mechanisms and conducting more comprehensive simulations that can provide deeper insights into the algorithm's behaviour under diverse computational conditions.

The limitations hardware diversity in the current research also suggests a promising direction for future work. Future research would develop more comprehensive simulation models that incorporate a wider range of edge computing hardware configurations. This would help establish the generalisability of BAMPP and explore hardware-specific optimisation strategies that could further enhance its performance.

Addressing the time frame constraints, future research would design extended simulation frameworks capable of capturing long-term performance characteristics and rare events. This would involve increasing simulation iterations and developing methodologies that can provide more statistically significant insights into the algorithm's behaviour over extended periods.

Building upon the current research, future work would continue to enhance BAMPP's reliability, particularly in scenarios showing higher variability. This will involve conducting in-depth analysis of performance inconsistencies and exploring potential modifications to reduce algorithmic variability. Expanding comparisons against a wider range of state-of-the-art algorithms will provide a more comprehensive understanding of BAMPP's capabilities.

© University of Pretoria

Ultimately, the goal is to enhance BAMPP's adaptive capabilities for handling dynamic and complex edge computing environments. This could involve incorporating advanced machine learning techniques for predictive scheduling, integrating BAMPP with emerging intelligent edge computing technologies, and developing sophisticated mechanisms for real-time adaptation to changing network conditions.

By pursuing these research directions, future work would continue to refine and expand the capabilities of BAMPP, ensuring its continued effectiveness and relevance in managing and optimising edge computing systems as the field evolves.

© University of Pretoria

REFERENCES

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[3] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surv. Tut.*, vol. 22, no. 2, pp. 869–904, 2nd Quart. 2020.

[4] S. Javanmardi, M. Shojafar, V. Persico, and A. Pescapè, "FPFTS: A joint fuzzy particle swarm optimization mobility-aware approach to fog task scheduling algorithm for Internet of Things devices," *Softw. Pract. Exper.*, vol. 51, no. 12, pp. 2519–2539, Dec. 2021.

[5] N. Hudson, H. Khamfroush, and D. E. Lucani, "QoS-aware placement of deep learning services on the edge with multiple service implementations," in *Proc. Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2021, pp. 1–8.

[6] T. L. Duc, R. G. Leiva, P. Casari, and P.-O. Östberg, "Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1–39, Sep. 2019.

[7] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, "A double deep Q-learning model

© University of Pretoria

for energy-efficient edge scheduling," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 739–749, Jan. 2019.

[8] K. Afachao, A. M. Abu-Mahfouz, and G. P. Hanke, "Efficient microservice deployment in the edge-cloud networks with policy-gradient reinforcement learning," *IEEE Access*, vol. 12, pp. 133 110–133 124, Sep. 2024.

[9] K. Afachao and A. M. Abu-Mahfouz, "Towards energy-efficient intelligent edge computing," in *Proc. Int. Conf. Elect. Comput. Energy Technol.*, Nov. 2023, pp. 1–6.

[10] K. Afachao and A. M. Abu-Mahfouz, "Running virtual services for the intelligent edge: A review," in *Proc. IEEE World AI IoT Congr.*, Jun. 2023, pp. 0204–0209.

[11] K. Afachao and A. M. Abu-Mahfouz, "An examination of virtualization technologies for enabling intelligent edge computing," in *Proc. Int. Conf. Sustain. Comput. Smart Syst.*, Jun. 2023, pp. 1612–1617.

[12] N. Dragoni *et al.*, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Cham, Switzerland: Springer, 2017, pp. 195–216.

[13] N. C. Mendonça, C. Box, C. Manolache, and L. Ryan, "The monolith strikes back: Why Istio migrated from microservices to a monolithic architecture," *IEEE Softw.*, vol. 38, no. 5, pp. 17–22, Sep./Oct. 2021.

[14] R. Vaño, I. Lacalle, P. Sowiński, R. S-Julián, and C. E. Palau, "Cloud-native workload orchestration at the edge: A deployment review and future directions," *Sensors*, vol. 23, no. 4, Feb. 2023.

[15] L. Sun, Y. Li, and R. A. Memon, "An open IoT framework based on microservices architecture," *China Commun.*, vol. 14, no. 2, pp. 154–162, Feb. 2017.

[16] S. Pallewatta, V. Kostakos, and R. Buyya, "Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments," in *Proc. 12th IEEE/ACM*

© University of Pretoria

Int. Conf. Utility Cloud Comput., Dec. 2019, pp. 71–81.

[17] B. Bermejo, C. Juiz, and C. Guerrero, "Virtualization and consolidation: A systematic review of the past 10 years of research on energy and performance," *J. Supercomput.*, vol. 75, no. 2, pp. 808–836, Feb. 2019.

[18] M. G. Xavier *et al.*, "Performance evaluation of container-based virtualization for high performance computing environments," in *Proc. 21st Euromicro Int. Conf. Parallel Distrib. Netw. Based Process.*, Feb. 2013, pp. 233–240.

[19] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open issues in scheduling microservices in the cloud," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 81–88, Sep./Oct. 2016.

[20] S. Hassan and R. Bahsoon, "Microservices and their design trade-offs: A self-adaptive roadmap," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2016, pp. 813–818.

[21] Y. Gan *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud and edge systems," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Apr. 2019, pp. 3–18.

[22] F. Tapia, M. A. Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, "From monolithic systems to microservices: A comparative study of performance," *Appl. Sci.*, vol. 10, no. 17, Aug. 2020.

[23] D. Gravanis, G. Kakarontzas, and V. Gerogiannis, "You don't need a microservices architecture (yet): Monoliths may do the trick," in *Proc. Eur. Symp. Softw. Eng.*, Nov. 2022, pp. 39 – 44.

[24] G. Blinowski, A. Ojdowska, and A. Przybyłek, "Monolithic vs. microservice architecture: A performance and scalability evaluation," *IEEE Access*, vol. 10, pp. 20 357–20 374, Feb. 2022.

[25] F. Tusa, S. Clayman, A. Buzachis, and M. Fazio, "Microservices and serverless functions—lifecycle, performance, and resource utilisation of edge based real-time IoT analytics," *Future Gener. Comput. Syst.*, vol. 155, pp. 204–218, Jun. 2024.

© University of Pretoria

[26] C. Allen, X. Li, A. S. Abdelfattah, T. Cerny, and D. Taibi, "Comparing cost and performance of microservices and serverless in AWS: EC2 vs lambda," in *Proc. Next Gener. Data Science*, Jun. 2024, pp. 60–72.

[27] Y. Mansouri and M. A. Babar, "A review of edge computing: Features and resource virtualization," *J. Parallel Distrib. Comput.*, vol. 150, pp. 155–183, Apr. 2021.

[28] M. De Donno, K. Tange, and N. Dragoni, "Foundations and evolution of modern computing paradigms: Cloud, IoT, edge, and fog," *IEEE Access*, vol. 7, pp. 150 936–150 948, Oct. 2019.

[29] I. Sittón-Candanedo, R. S. Alonso, J. M. Corchado, S. Rodríguez-González, and R. Casado-Vara, "A review of edge computing reference architectures and a new global edge proposal," *Future Gener. Comput. Syst.*, vol. 99, pp. 278–294, Oct. 2019.

[30] H. Tabatabaee Malazi *et al.*, "Dynamic service placement in multi-access edge computing: A systematic literature review," *IEEE Access*, vol. 10, pp. 32 639–32 688, Mar. 2022.

[31] A. Samanta and J. Tang, "Dyme: Dynamic microservice scheduling in edge computing enabled IoT," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6164–6174, Jul. 2020.

[32] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT using docker and edge computing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 118–123, Sep. 2018.

[33] A. Ali-Eldin, B. Wang, and P. Shenoy, "The hidden cost of the edge: A performance comparison of edge and cloud latencies," in *Proc. Int. Conf. High Perform. Comput. Netwo. Storage Anal.*, Nov. 2021, pp. 1–15.

[34] V. Scoca, A. Aral, I. Brandic, R. D. Nicola, and R. B. Uriarte, "Scheduling latency-lensitive applications in edge computing," in *Proc. 8th Int. Conf. Cloud Comput. Services Sci.*, Mar. 2018, pp. 158–168.

© University of Pretoria

[35] A. Aral, I. Brandic, R. B. Uriarte, R. De Nicola, and V. Scoca, "Addressing application latency requirements through edge scheduling," *J. Grid Comput.*, vol. 17, no. 4, pp. 677 – 698, Dec. 2019.

[36] C. Caiazza, S. Giordano, V. Luconi, and A. Vecchio, "Edge computing vs centralized cloud: Impact of communication latency on the energy consumption of LTE terminal nodes," *Comput. Commun.*, vol. 194, pp. 213–225, Oct. 2022.

[37] E. Ahvar, A. Orgerie, and A. Lebre, "Estimating energy consumption of cloud, fog, and edge computing infrastructures," *IEEE Trans. Sustain. Comput.*, vol. 7, no. 2, pp. 277–288, Mar. 2022.

[38] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7457–7469, Aug. 2020.

[39] M. Żyliński, A. Nassibi, I. Rakhmatulin, A. Malik, C. M. Papavassiliou, and D. P. Mandic, "Deployment of artificial intelligence models on edge devices: A tutorial brief," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 71, no. 3, pp. 1738–1743, Mar. 2024.

[40] S. S. Gill *et al.*, "Edge AI: A taxonomy, systematic review and future directions," *Cluster Comput.*, vol. 28, no. 1, pp. 1–53, Oct. 2025.

[41] P. Singh, M. J. Beliatis, and M. Presser, "Enabling edge-driven dataspace integration through convergence of distributed technologies," *Internet Things*, vol. 25, p. 101087, Apr. 2024.

[42] F. Chen, S. Li, J. Han, F. Ren, and Z. Yang, "Review of lightweight deep convolutional neural networks," *Arch. Comput. Methods Eng.*, vol. 31, no. 4, pp. 1915–1937, May 2024.

[43] M. Mukherjee, R. Matam, C. X. Mavromoustakis, H. Jiang, G. Mastorakis, and M. Guo, "Intelligent edge computing: Security and privacy challenges," *IEEE Commun. Mag.*, vol. 58, no. 9, pp. 26–31, Sep. 2020.

© University of Pretoria

[44] W. Li, H. Hacid, E. Almazrouei, and M. Debbah, "A comprehensive review and a taxonomy of edge machine learning: Requirements, paradigms, and techniques," *AI*, vol. 4, no. 3, pp. 729–786, Sep. 2023.

[45] J. L. Ribeiro, M. Figueredo, A. Araujo, N. Cacho, and F. Lopes, "A microservice based architecture topology for machine learning deployment," in *Proc. IEEE Int. Smart Cities Conf. (ISC2)*, Oct. 2019, pp. 426–431.

[46] G. D. Gasperis, G. D. Penna, and S. D. Facchini, "A microservices architecture for machine learning assisted decision support in a real-time field sensors environment (short paper)," in *Eur. Conf. Softw. Archit.*, 2021.

[47] G. Ortiz *et al.*, "Real-time context-aware microservice architecture for predictive analytics and smart decision-making," *IEEE Access*, vol. 7, pp. 183 177–183 194, Dec. 2019.

[48] P. Keshavarz Haddadha, M. H. Rezvani, M. MollaMotalebi, and A. Shankar, "Machine learning methods for service placement : A systematic review," *Artif. Intell. Rev.*, vol. 57, no. 3, Feb. 2024.

[49] J. Lv, M. Wei, and Y. Yu, "A container scheduling strategy based on machine learning in microservice architecture," in *Proc. IEEE Int. Conf. Serv. Comput. (SCC)*, Jul. 2019, pp. 65–71.

[50] W. Lv *et al.*, "Microservice deployment in edge computing based on deep Q learning," *IEEE Trans. Parallel and Distrib. Syst.*, vol. 33, no. 11, pp. 2968–2978, Nov. 2022.

[51] L. Chen *et al.*, "IoT microservice deployment in edge-cloud hybrid environment using reinforcement learning," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12 610–12 622, Aug. 2021.

[52] M. Hu *et al.*, "Joint optimization of microservice deployment and routing in edge via multi-objective deep reinforcement learning," *IEEE Trans. Netw. Service Manage.*, vol. 21, no. 4, pp. 1–1, Aug. 2024.

© University of Pretoria

[53] K. Peng *et al.*, "Delay-aware optimization of fine-grained microservice deployment and routing in edge via reinforcement learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 6, pp. 6024–6037, Nov. - Dec. 2024.

[54] G. Tong, C. Meng, S. Song, M. Pan, and Y. Yu, "GMA: Graph multi-agent microservice autoscaling algorithm in edge-cloud environment," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2023, pp. 393–404.

[55] C. Wang *et al.*, "Dependency-aware microservice deployment for edge computing: A deep reinforcement learning approach with network representation," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 14 737–14 753, Dec. 2024.

[56] S. Chen *et al.*, "Graph neural network aided deep reinforcement learning for microservice deployment in cooperative edge computing," *IEEE Trans. Services Comput.*, vol. 17, no. 3, pp. 1–16, Jun. 2024.

[57] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 712–725, Sep./Oct. 2019.

[58] K. Fu *et al.*, "QoS-aware and resource efficient microservice deployment in cloud-edge continuum," in *Proc. IEEE Int. Parallel and Distrib. Process. Symp. (IPDPS)*, May 2021, pp. 932–941.

[59] C. Wu, Q. Peng, Y. Xia, Y. Jin, and Z. Hu, "Towards cost-effective and robust AI microservice deployment in edge computing environments," *Future Gener. Comput. Syst.*, vol. 141, pp. 129 – 142, Apr. 2023.

[60] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Mar. 2021.

© University of Pretoria

[61] Y. Yu, J. Liu, and J. Fang, "Online microservice orchestration for IoT via multiobjective deep reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 17 513–17 525, Mar. 2022.

[62] M. Golec *et al.*, "ATOM: AI-powered sustainable resource management for serverless edge computing environments," *IEEE Trans. Sustain. Comput.*, pp. 1–13, Dec. 2023.

[63] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: An overview," in *Proc. SAI Intell. Syst. Conf.* Springer, Jun. 2018, pp. 426–440.

[64] R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya, "iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments," *J. Syst. Softw.*, vol. 190, pp. 111–351, Aug. 2022.

[65] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, arXiv:1707.06347.

© University of Pretoria

APPENDIX A MICROSERVICES SIMULATION PARAMETERS

Table A.1, Table A.2, and Table A.3 provide detailed configuration settings for microservices in three different applications: CHM, ECOM, and DRN. These tables provide the resource requirements and interdependencies of microservices within each application, serving as the foundation for the simulation and optimisation discussed in the dissertation.

Table A.1. The configuration settings for CHM microservices.

Microservice	CPU (GHz)	RAM (MB)	Usage rate (MB/s)	Dependencies
1	(0.1 0.3)	20	(2 16)	
2	(0.1 0.3)	5	(2 16)	[1]
3	(0.1 0.3)	10	(2 16)	[2]
4	(0.1 0.3)	15	(2 16)	[2]

Table A.2. The configuration settings for ECOM microservices.

Microservice	CPU (GHz)	RAM (MB)	Usage rate (MB/s)	Dependencies
1	(0.1 0.3)	150	(2 32)	
2	(0.1 0.3)	80	(2 32)	[1]
3	(0.1 0.3)	100	(2 32)	[1]
4	(0.1 0.3)	100	(2 32)	[2]
5	(0.1 0.3)	60	(2 32)	[3]
6	(0.1 0.3)	40	(2 32)	[4]
7	(0.1 0.5)	60	(2 32)	[4]

Continued on next page

© University of Pretoria

Table A.2 – *Continued from previous page*

Microservice	**CPU (GHz)**	**RAM (MB)**	**Usage rate (MB/s)**	**Dependencies**
8	(0.1 0.3)	90	(2 32)	[5]
9	(0.1 0.5)	60	(2 32)	[6]
10	(0.1 0.5)	100	(2 32)	[6]
11	(0.1 0.3)	70	(2 32)	[7,8]
12	(0.1 0.3)	75	(2 32)	[10,11]
13	(0.1 0.5)	100	(2 32)	[9]
14	(0.1 0.3)	100	(2 32)	[9,12]
15	(0.1 0.5)	90	(2 32)	[13]
16	(0.1 0.5)	20	(2 32)	[14]
17	(0.1 0.5)	80	(2 32)	[15, 16]
18	(0.1 0.3)	20	(2 32)	[17]
19	(0.1 0.5)	100	(2 32)	[17]
20	(0.1 0.3)	90	(2 32)	[17]
21	(0.1 0.3)	120	(2 32)	[18,19,20]
22	(0.1 0.3)	100	(2 32)	[21]
23	(0.1 0.3)	150	(2 32)	[21]
24	(0.1 0.3)	100	(2 32)	[21]
25	(0.1 0.3)	150	(2 32)	[22,23,24]
26	(0.1 0.5)	100	(2 32)	[25]
27	(0.1 0.3)	60	(2 32)	[25]
28	(0.1 0.3)	100	(2 32)	[25]
29	(0.1 0.3)	80	(2 32)	[26,27,28]
30	(0.1 0.3)	80	(2 32)	[29]
31	(0.1 0.3)	80	(2 32)	[29]
32	(0.1 0.3)	100	(2 32)	[29]
33	(0.1 0.3)	80	(2 32)	[32]
34	(0.1 0.3)	100	(2 32)	[33]

© University of Pretoria

Table A.3. The configuration settings for DRN microservices.

Microservice	CPU (GHz)	RAM (MB)	Usage rate (MB/s)	Dependencies
1	(0.1 0.3)	50	(2 32)	
2	(0.1 0.3)	100	(2 32)	[1]
3	(0.1 0.3)	80	(2 32)	[1,4]
4	(0.1 0.3)	60	(2 32)	[3]
5	(0.1 0.3)	120	(2 32)	[4,6]
6	(0.1 0.5)	90	(2 32)	[5]
7	(0.1 0.3)	150	(2 32)	[5,8]
8	(0.1 0.3)	70	(2 32)	[7]
9	(0.1 0.5)	180	(2 32)	[7,10]
10	(0.1 0.3)	110	(2 32)	[9]
11	(0.1 0.3)	40	(2 32)	[9,12]
12	(0.1 0.5)	80	(2 32)	[10,11]
13	(0.1 0.3)	60	(2 32)	[11]
14	(0.1 0.3)	90	(2 32)	[13]
15	(0.1 0.5)	120	(2 32)	[14]
16	(0.1 0.5)	200	(2 32)	[15]
17	(0.1 0.3)	300	(2 32)	[16, 18]
18	(0.1 0.5)	180	(2 32)	[17]
19	(0.1 0.5)	100	(2 32)	[17]
20	(0.1 0.3)	60	(2 32)	[19]
21	(0.1 0.3)	80	(2 32)	[20]

© University of Pretoria

APPENDIX B NUMERICAL RESULTS

Table B.1, Table B.2, Table B.3, Table B.4, Table B.5, and Table B.6 provide a detailed comparison of the performance metrics and reward scores for the different optimisation methods across the three microservices applications.

Table B.1. The average performance metrics for CHM application.

Metric	ILP	PPO	A2C	BAMPP
Network Usage (Mbps)	5.522	5.472	4.2158	5.026
Execution time (s)	2.984	22.254	10.873	7.427
Migration delay (s)	2.995	2.995	2.995	2.995
Energy consumption (kWh)	177.400	170	170	166

Table B.2. The average performance metrics for DRN application.

Metric	ILP	PPO	A2C	BAMPP
Network Usage (Mbps)	92.9	29.129	50.291	42.186
Execution time (s)	2.984	22.254	10.873	7.427
Migration delay (s)	6.494	6.494	6.334	6.227
Energy consumption (kWh)	171.600	166.400	168.300	157.600

© University of Pretoria

Table B.3. The average performance metrics for ECOM application.

Metric	ILP	PPO	A2C	BAMPP
Network Usage (Mbps)	107.092	13.626	12.129	8.869
Execution time (s)	2.984	22.254	10.873	7.427
Migration delay (s)	4.707	4.853	4.853	4.823
Energy consumption (kWh)	206.400	202.650	188.500	172.800

Table B.4. The average reward scores and standard deviation for CHM application.

Metric	PPO	A2C	BAMPP
Reward	6138.8	22775.3	24532.7
Standard deviation	445.751	189.375	222.538

Table B.5. The average reward scores and standard deviation for DRN application.

Metric	PPO	A2C	BAMPP
Reward	1153.175	24842.5	24994.075
Standard deviation	280.519	62.189	38.544

Table B.6. The average reward scores and standard deviation for ECOM application.

Metric	PPO	A2C	BAMPP
Reward	2056.75	24943.25	24987.25
Standard deviation	110.496	11.906	18.198

© University of Pretoria